



Next Steps with Blockchain Technology

Eliezer Kanal & Gabriel Somlo

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213



Copyright 2018 Carnegie Mellon University. All Rights Reserved.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

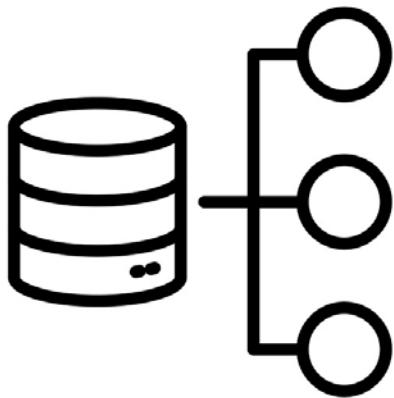
NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

Carnegie Mellon® and CERT® are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University. DM18-1016

Previous models of computing

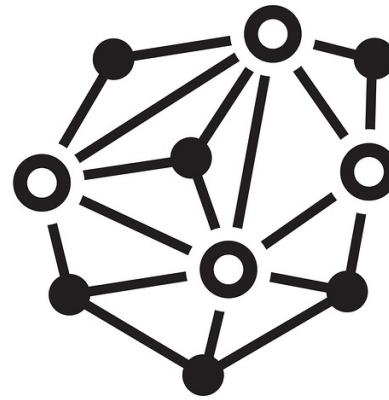


Data Storage:
Database



Program Execution:
Local

Blockchain

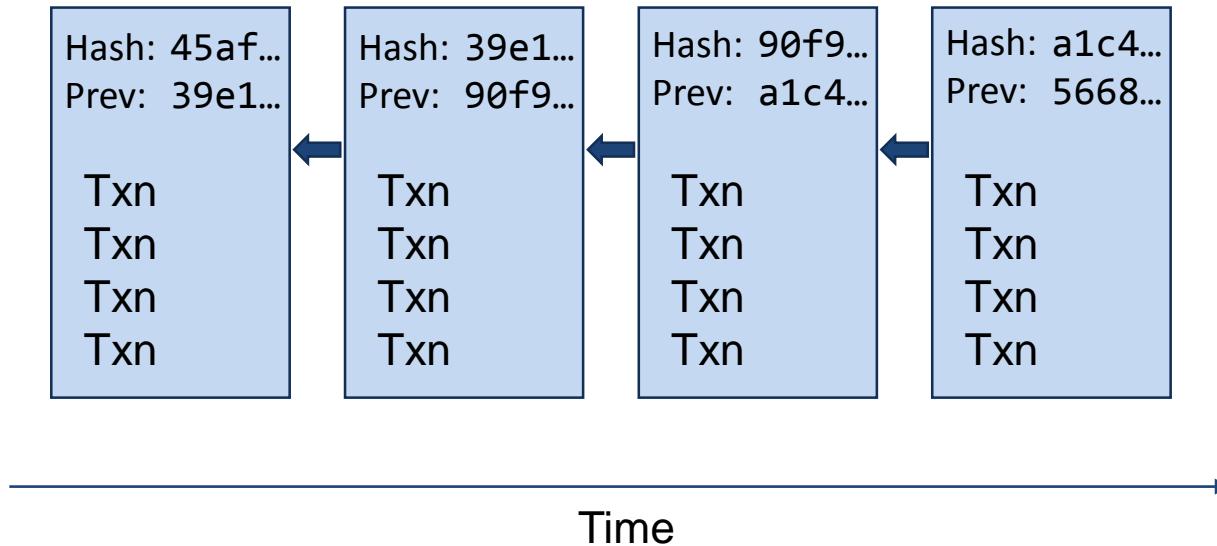


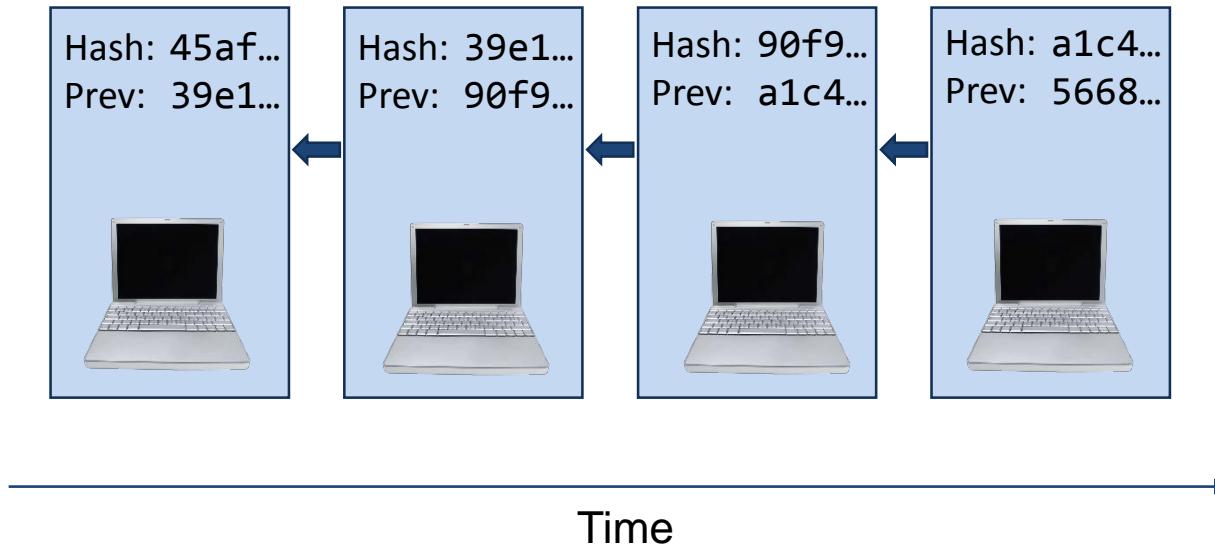
Data Storage:

Blockchain or Network

Program Execution:

Network





<u>Candidate</u>	<u>Votes</u>
Bob	0
Jim	0
Frank	0

State: 1

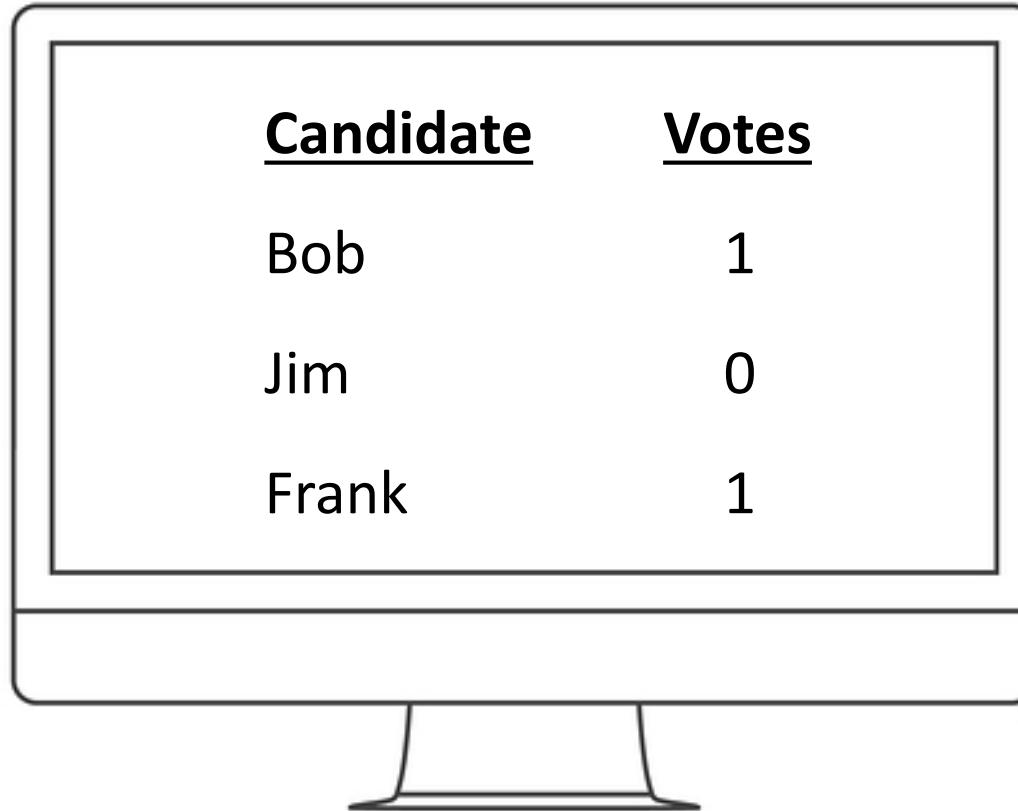


Bob: 1 vote



Frank: 1 vote

<u>Candidate</u>	<u>Votes</u>
Bob	0
Jim	0
Frank	0



State: 1

<u>Candidate</u>	<u>Votes</u>
Bob	0
Jim	0
Frank	0

State: 2

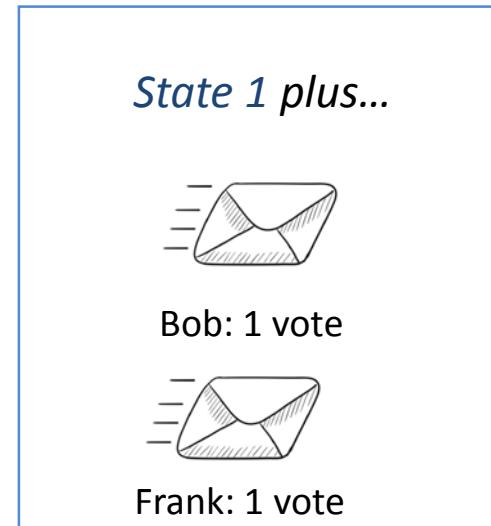
<u>Candidate</u>	<u>Votes</u>
Bob	1
Jim	0
Frank	1

Equivalent to:

State: 1

<u>Candidate</u>	<u>Votes</u>
Bob	0
Jim	0
Frank	0

State: 2



Blockchain: Executive Summary

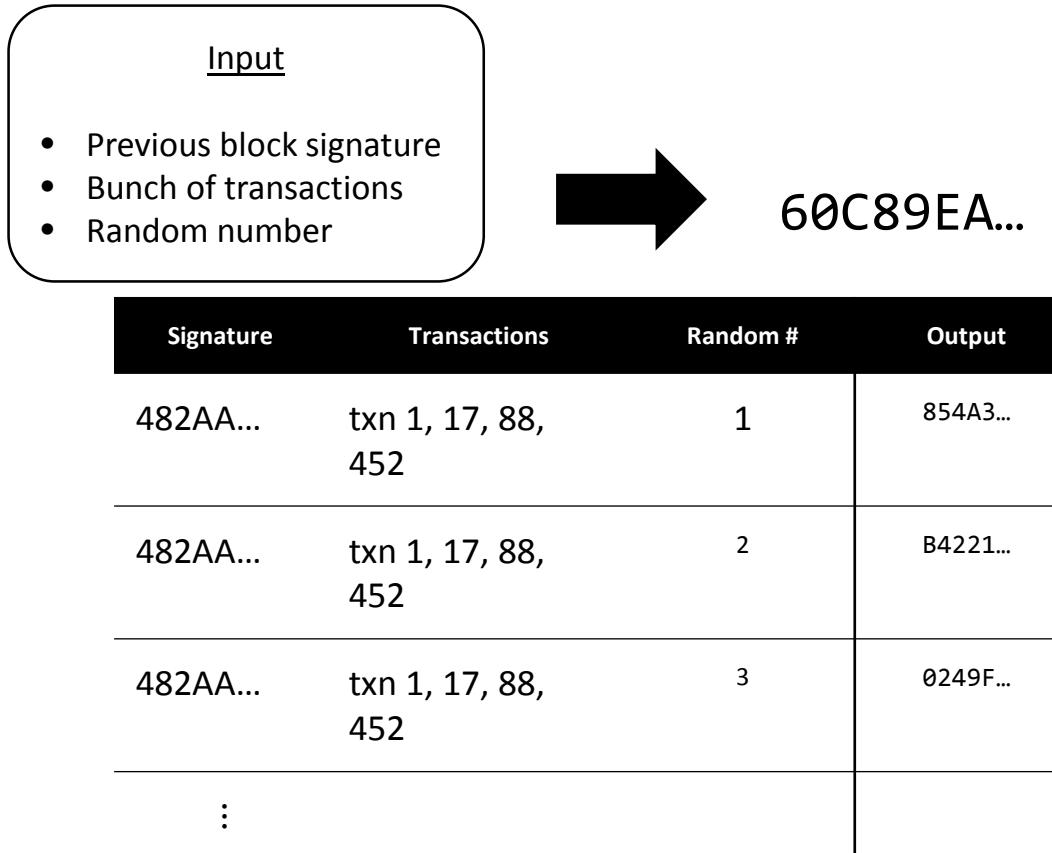
Pros:

- Authentication built-in
- Easy to audit history
- Easy to detect data manipulation
- Very difficult to disrupt

Cons:

- Proof-of-work very inefficient
- State updates are slow
- Best for simple computations

Bitcoin: Mining



Block #509169

Summary

Number Of Transactions	1915
Output Total	10,289.28130284 BTC
Estimated Transaction Volume	1,818.68925455 BTC
Transaction Fees	0.4893378 BTC
Height	509169 (Main Chain)
Timestamp	2018-02-14 15:16:59
Received Time	2018-02-14 15:16:59
Relayed By	58COIN
Difficulty	2,874,674,234,415.94
Bits	392292856
Size	1132.416 kB
Weight	3992.574 kWU
Version	0x20000000
Nonce	1856980081
Block Reward	12.5 BTC

Hashes

Hash	000000000000000000000000000000002c4b94355945eea353bc720c58a73c2b8593f489550cb3
Previous Block	00000000000000000000000000000001d620a2e3ad126ec5038bf42343c419eb6fcdf7240a471
Next Block(s)	
Merkle Root	3ad680735c45cc62b1ea6b7efeb34fb2a2680c5e8280354c45f7fa03c9137e2

Transactions

[ab0da64ea834fd2acb81eb081d8103c9e31fd14a7d055f2ce2718c59dd4fa5df](#)

2018-02-14 15:16:59

No Inputs (Newly Generated Coins)



[14DTuAUh87cwRsbU1z6W8hZY6FnEkpfLS](#)
Unable to decode output address

12.9893378 BTC
0 BTC

[12.9893378 BTC](#)

[4feb8981da942b10a2a3b84003fb1c1d78c8f192cd2747e43ae552ed237f267d](#)

2018-02-14 15:16:59

[1H6ZZpRmMnrw8ytepV3BYwMjYYnEkWDqVP](#)



[12PaHlRJBmvJYmTpZ32Pswf8eYbKcAE131](#)
[1GpqR4vsdvEfgtNyIUrDfDLTBjvnsetX](#)
[1H6ZZpRmMnrw8ytepV3BYwMjYYnEkWDqVP](#)

0.4983 BTC
0.1495 BTC
5.01651602 BTC

[5.66431602 BTC](#)

1H6ZZpRmMnrw8ytepV3BYwMjYYnEkWDqVP

12PaHirJBmvJYmTpZ32Pswf8eYbKcAE131
1GpqR4vsdvEfgtNylUrDrfDLTBjvnsentX
1H6ZZpRmMnrw8ytepV3BYwMjYYnEkWDqVP

0.4983 BTC
0.1495 BTC
5.01651602 BTC

5.66431602 BTC

Estimated Transaction Volume

1.818,68925455 BTC

Next Block(s)

Transaction

Height

Timestamp

Received Time

2018-02-14 15:16:59

Relay

Difficulty

Bits

Size

1132,416 kB

Number Of Transactions

1915

60354c45f7ffa03c9137e2

Nonce

1858980081
,

Difficulty

2,874,674,234,415.94

Block Reward

12.5 BTC

Hash

00000000000000000000000000002c4b94355945eea353bc720c58a73c2b8593f489550cb3

Previous Block

00000000000000000000000000001d620a2e3ad126ec5038bf42343c419eb6fcdf7240a471

Block Reward

12.5 BTC

12.9893378 BTC

4feb89

1H6ZZ

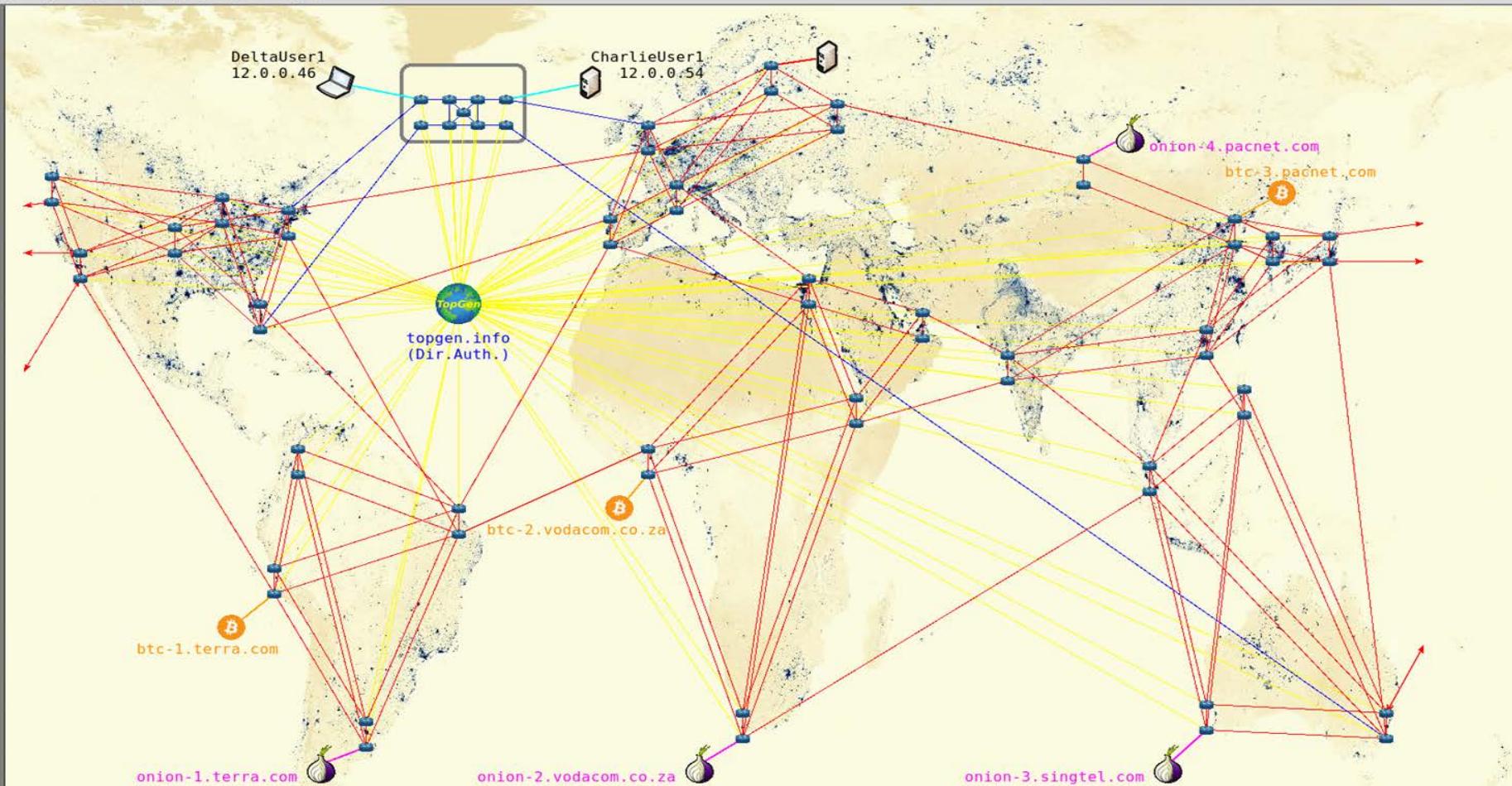
18-02-14 15:16:59

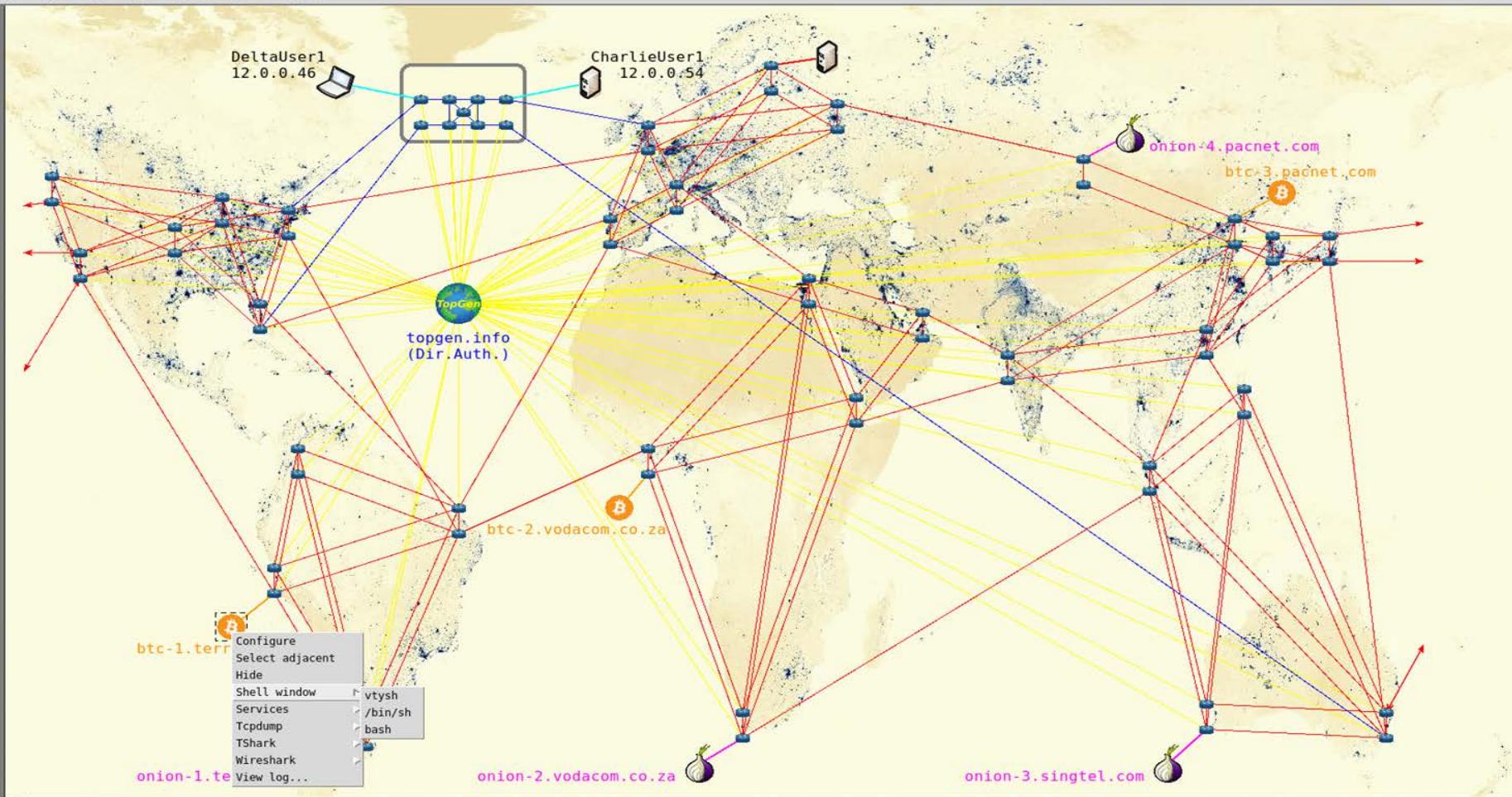
0.4983 BTC

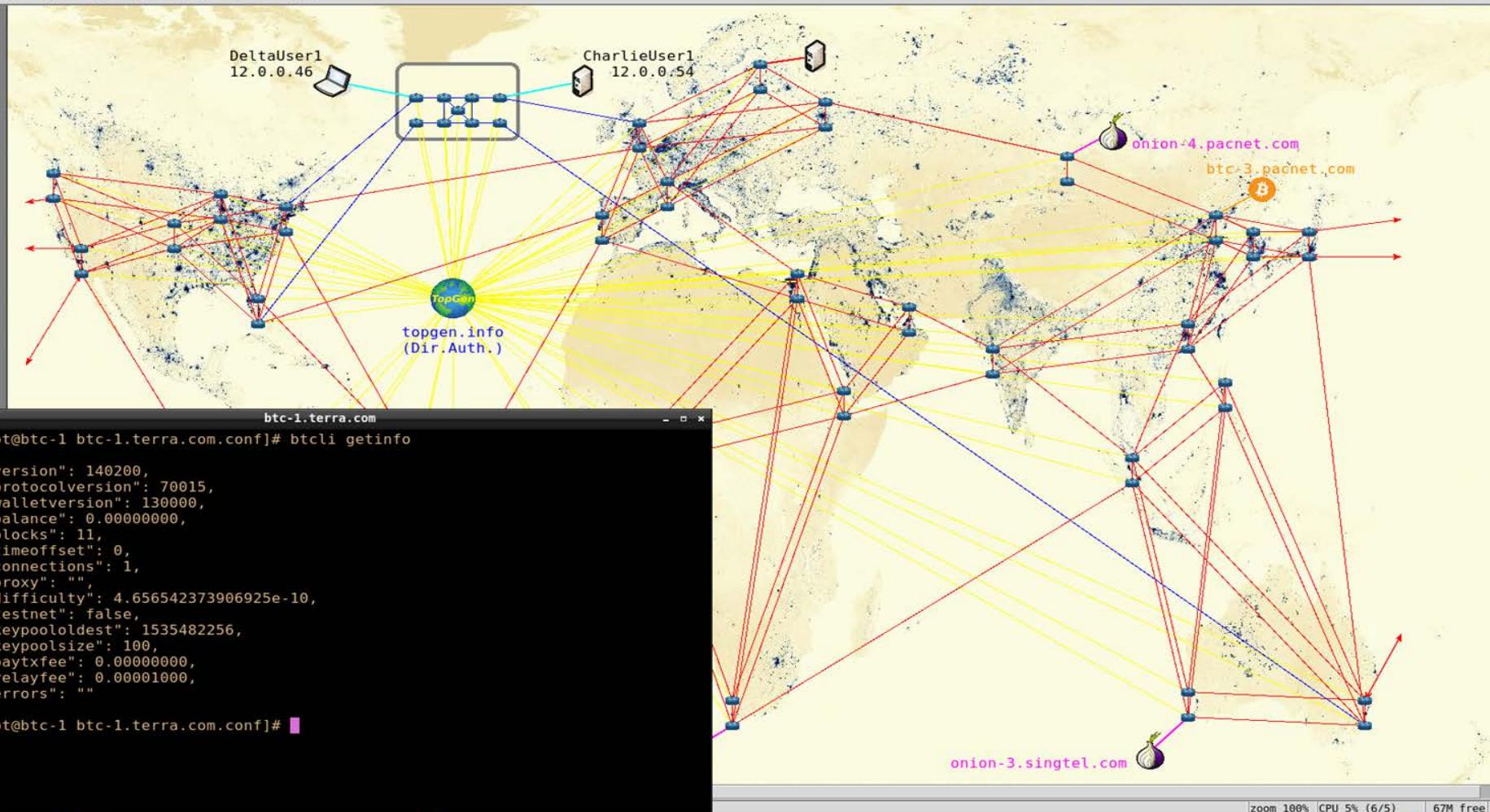
0.1495 BTC

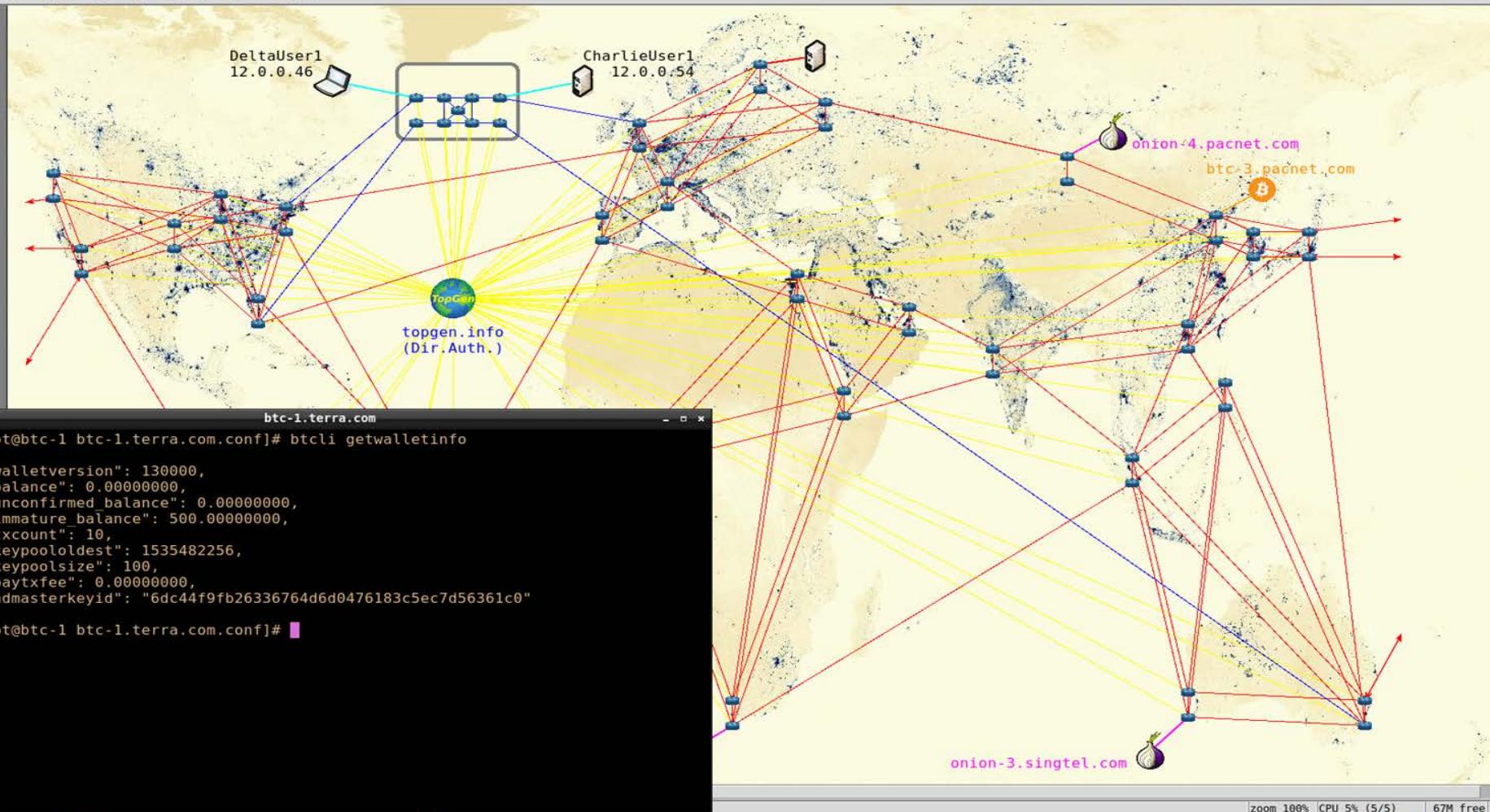
5.01651602 BTC

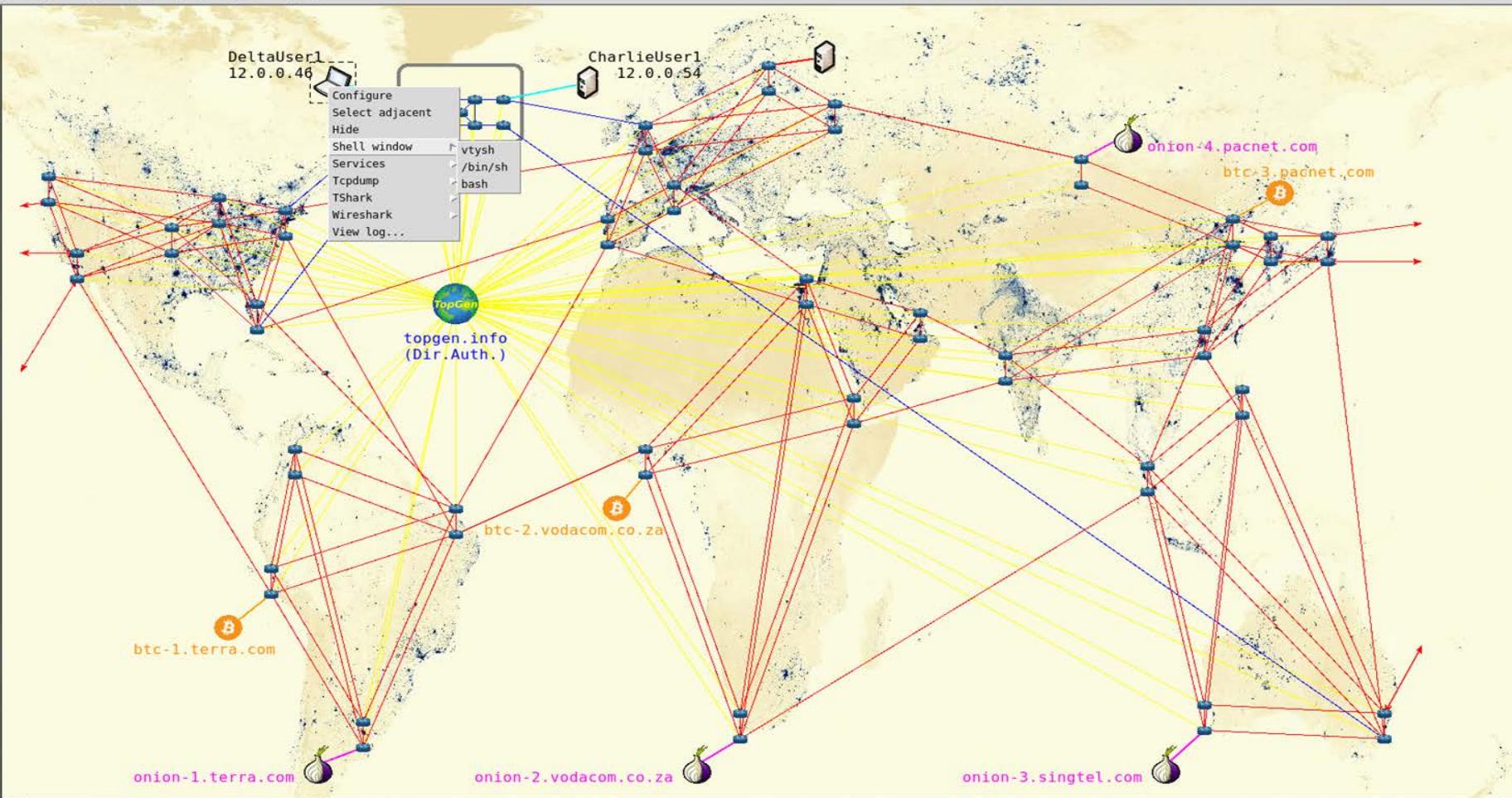
5.66431602 BTC

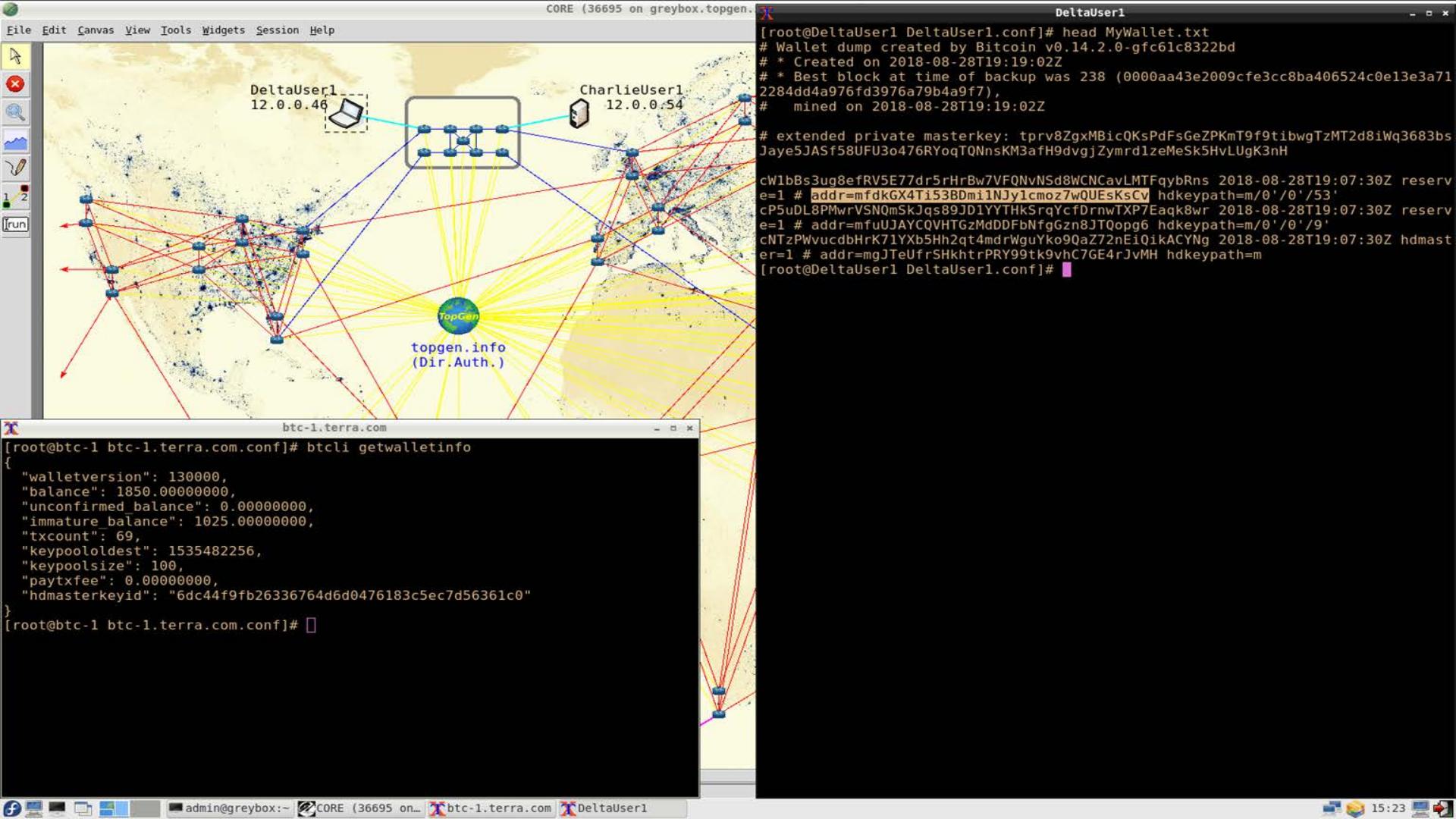


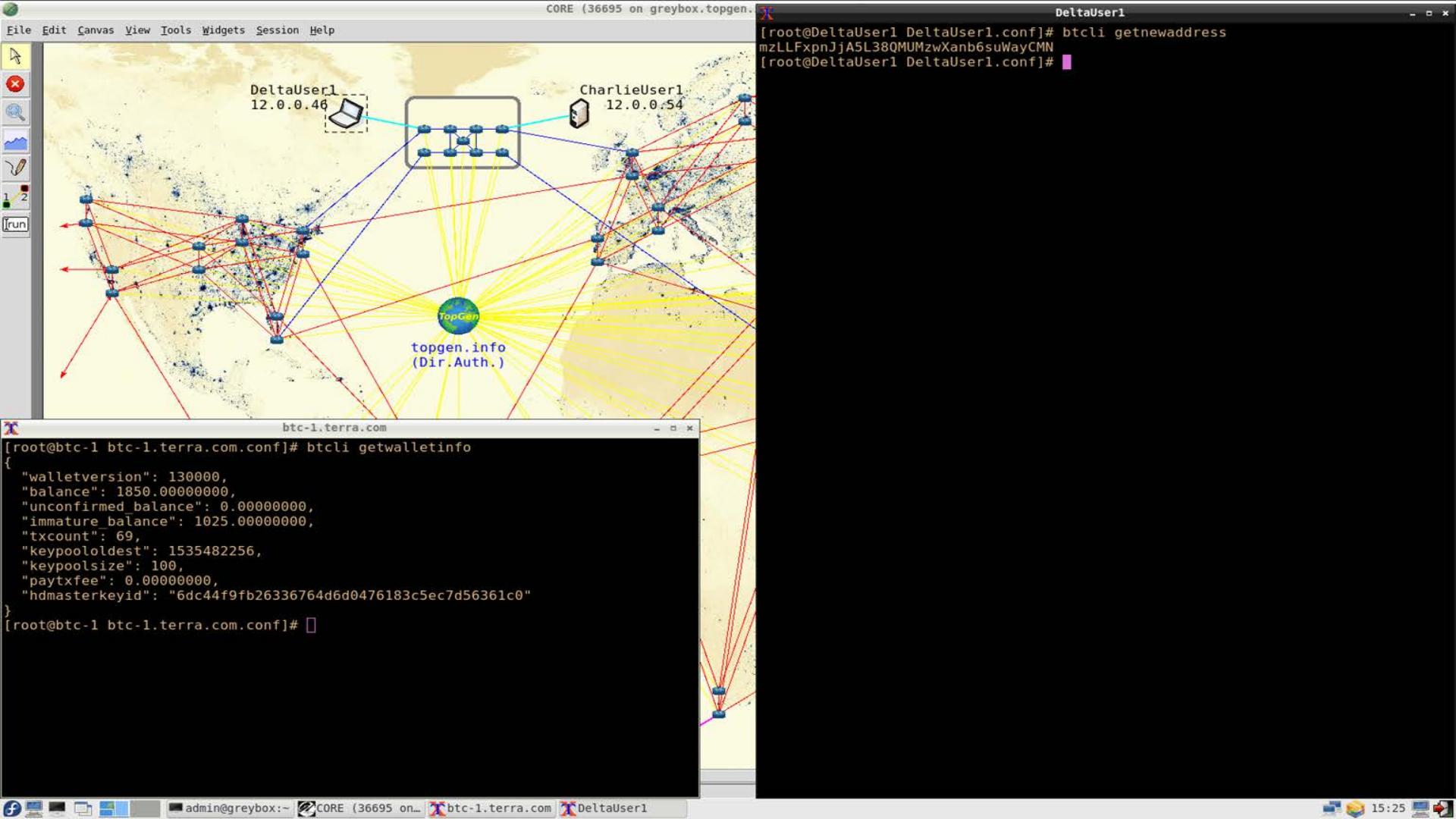


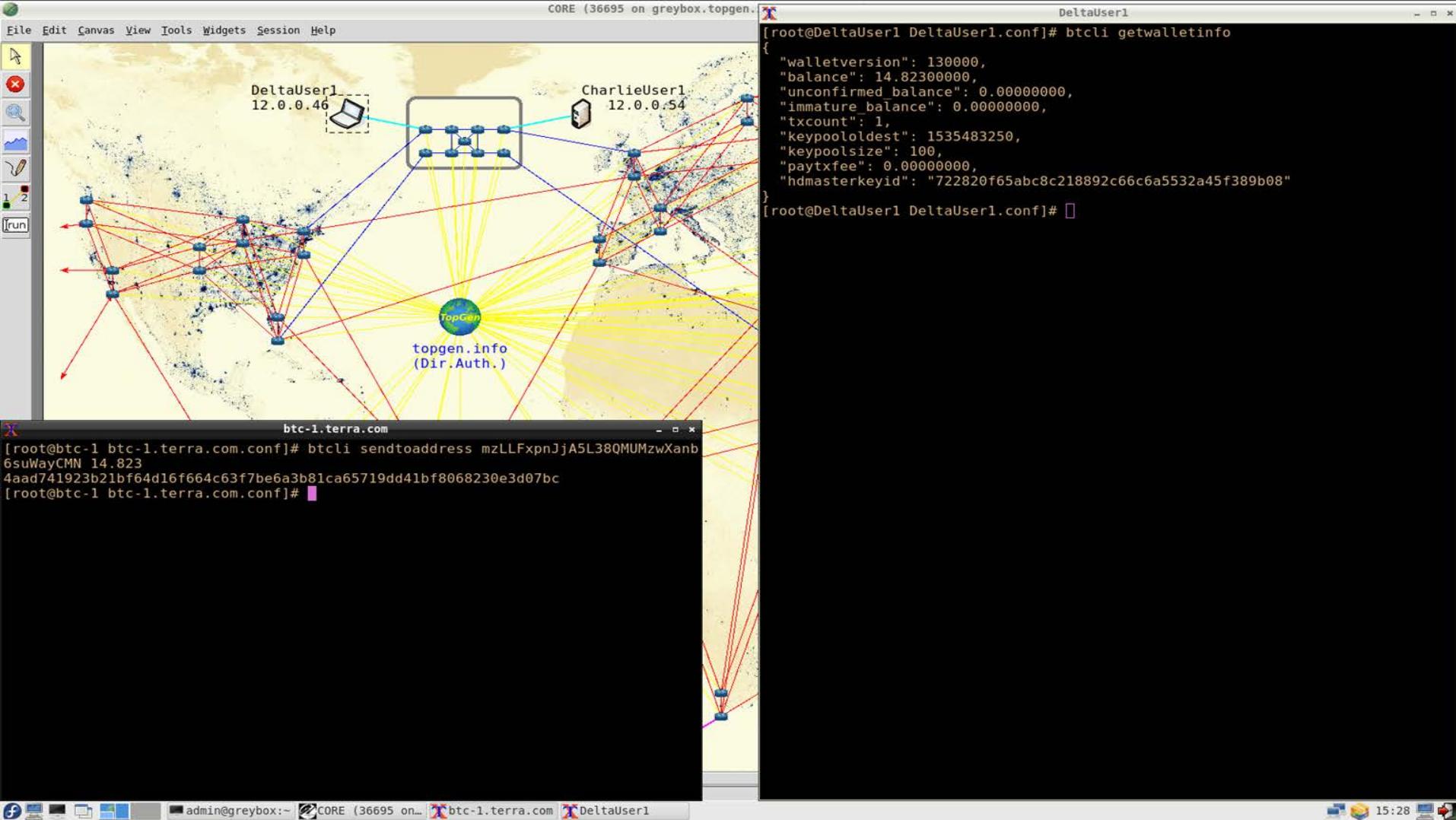


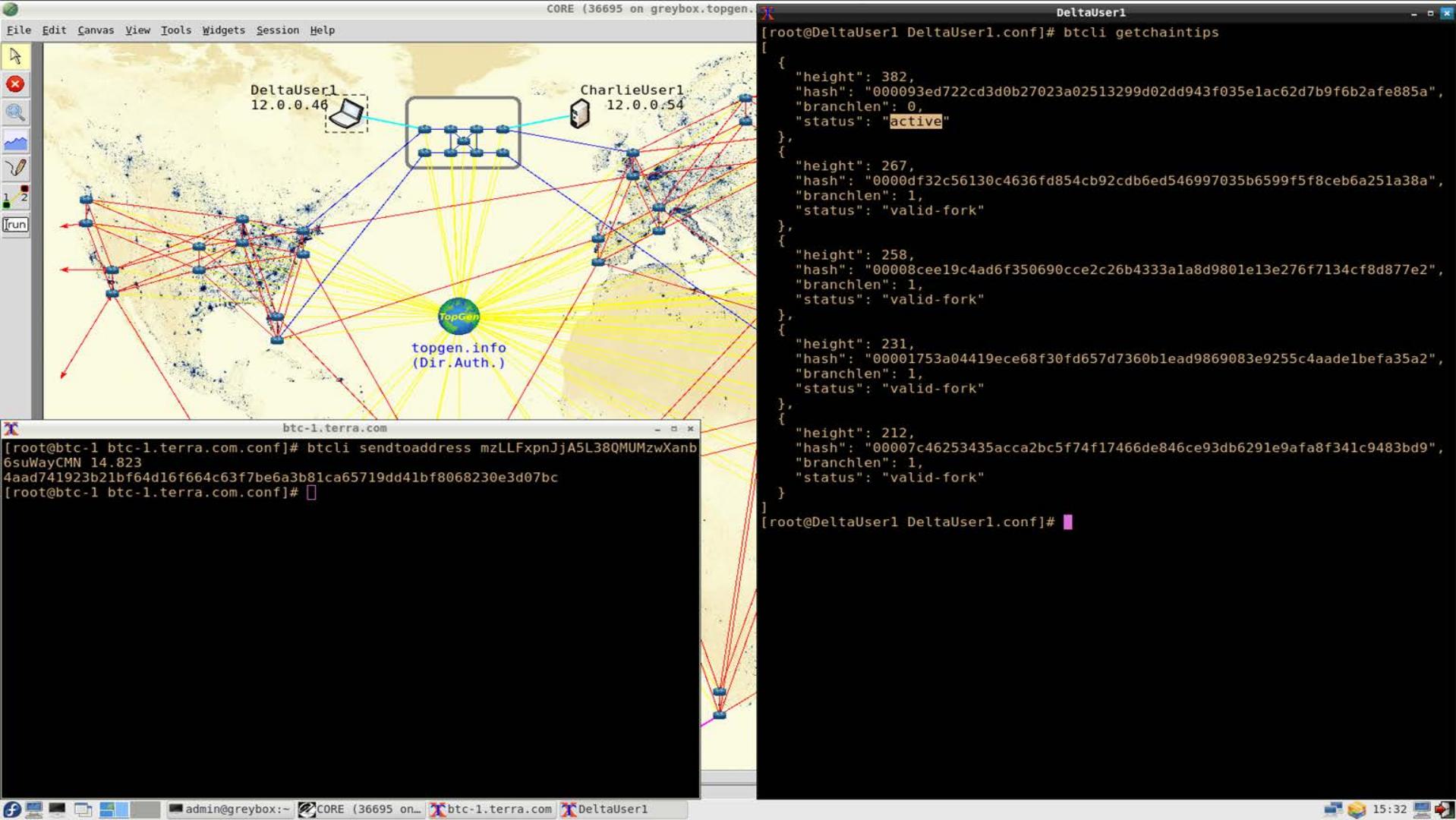


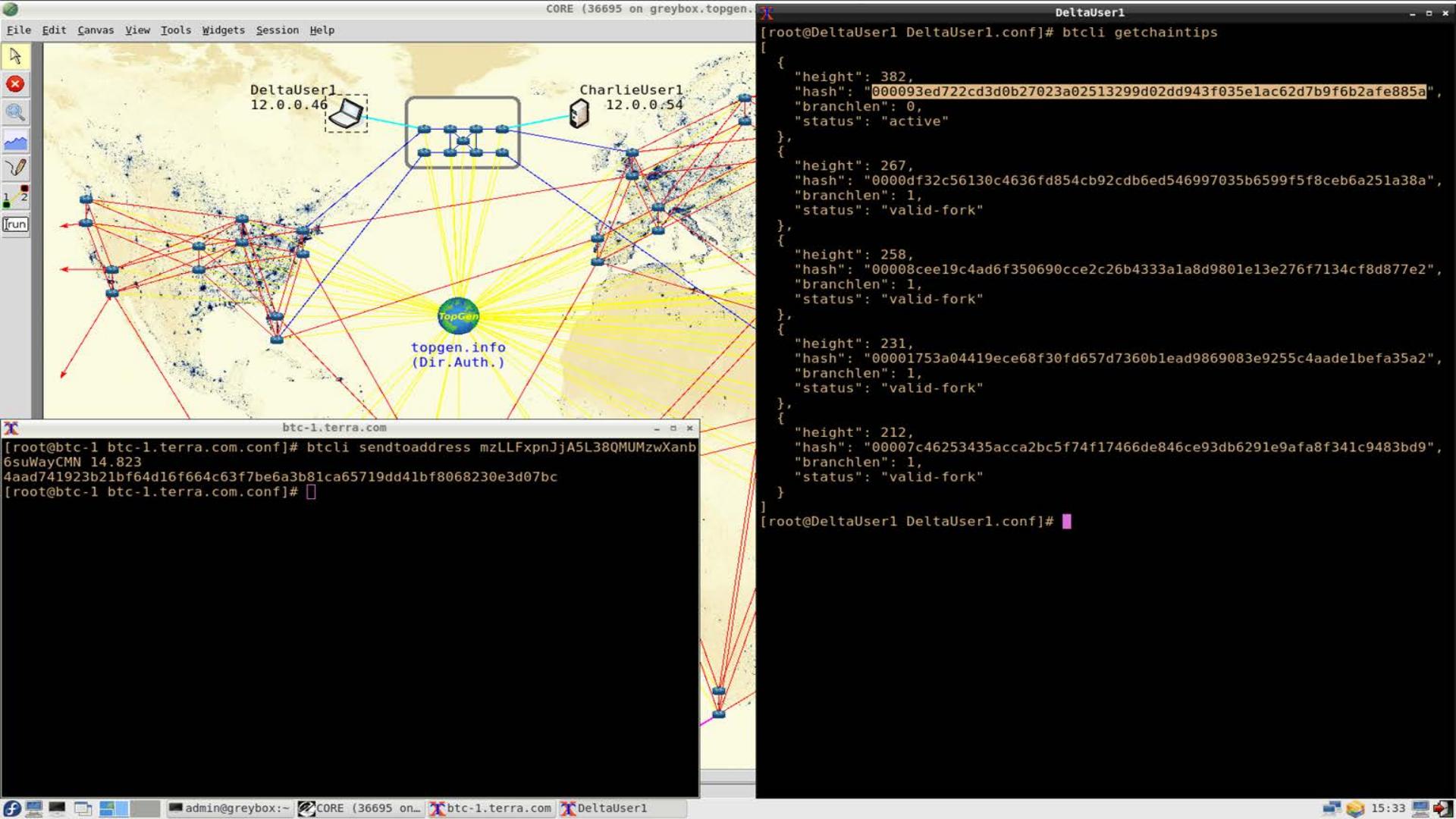


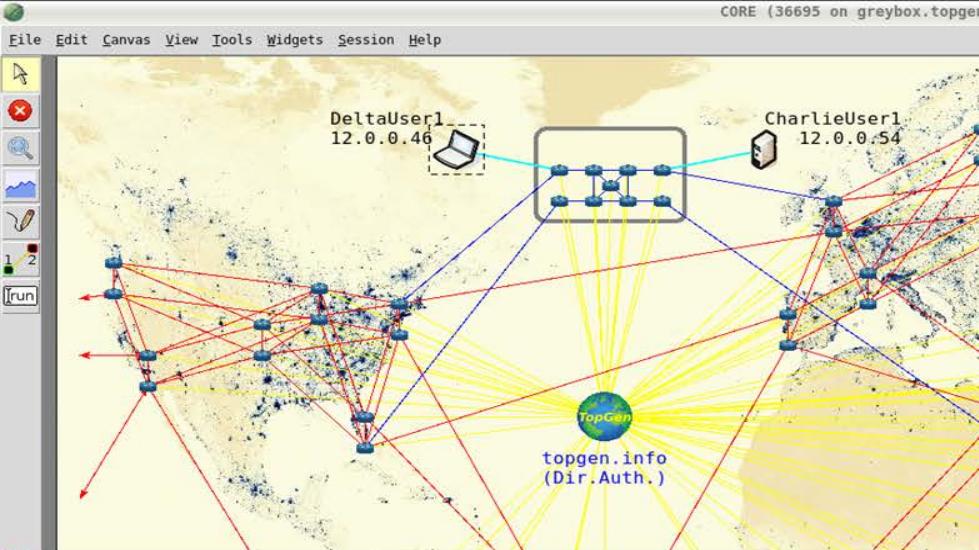




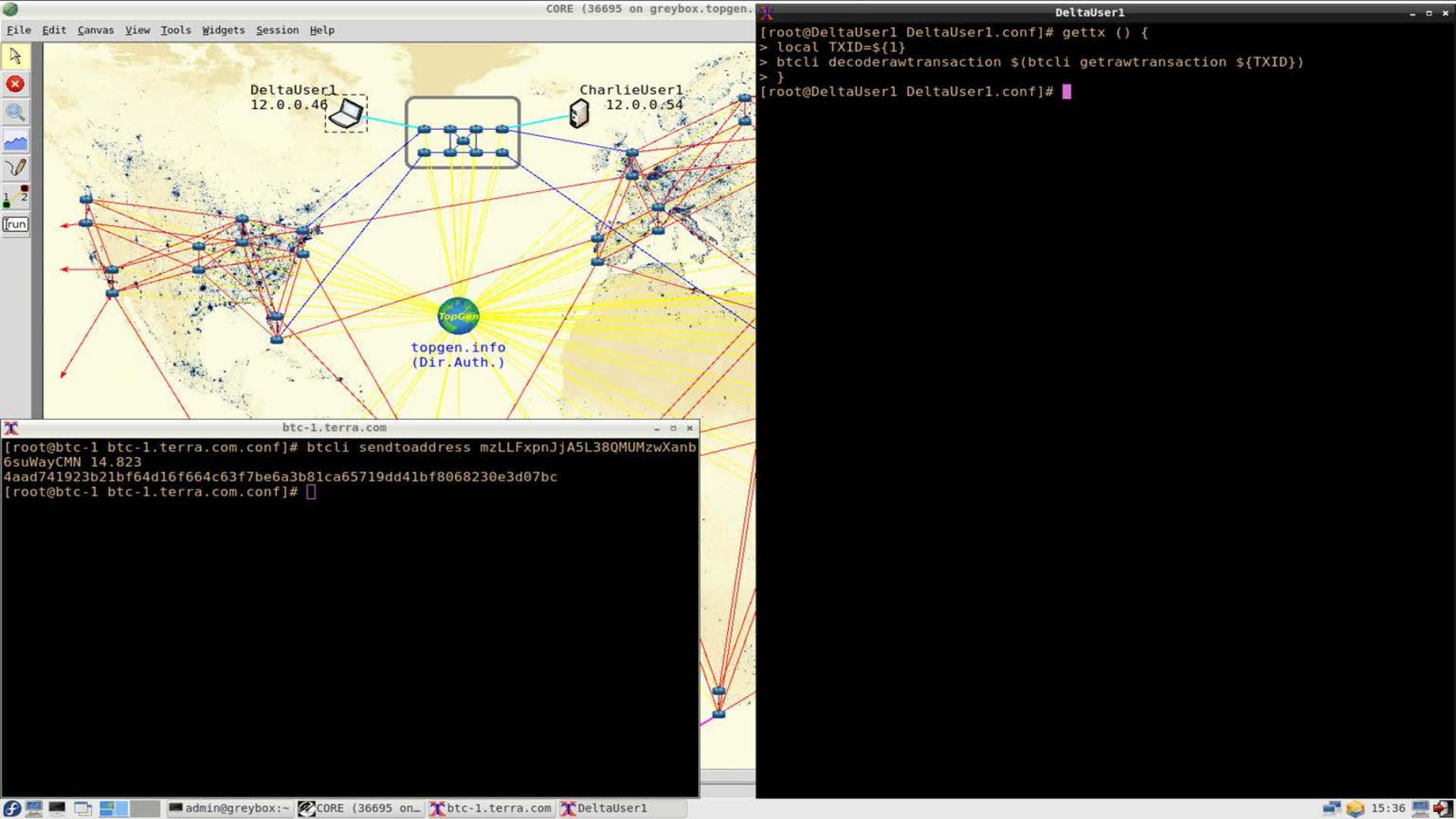


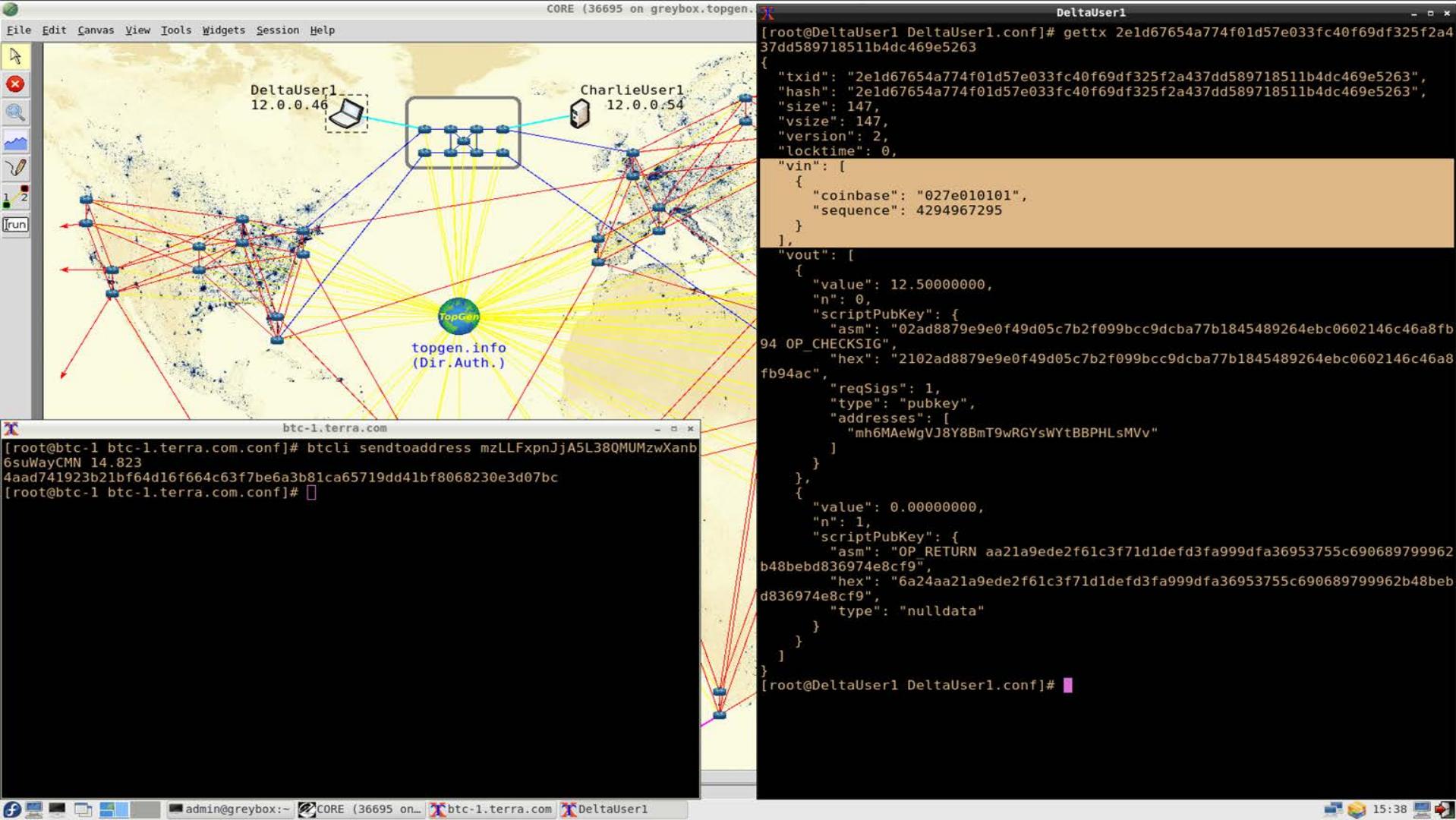


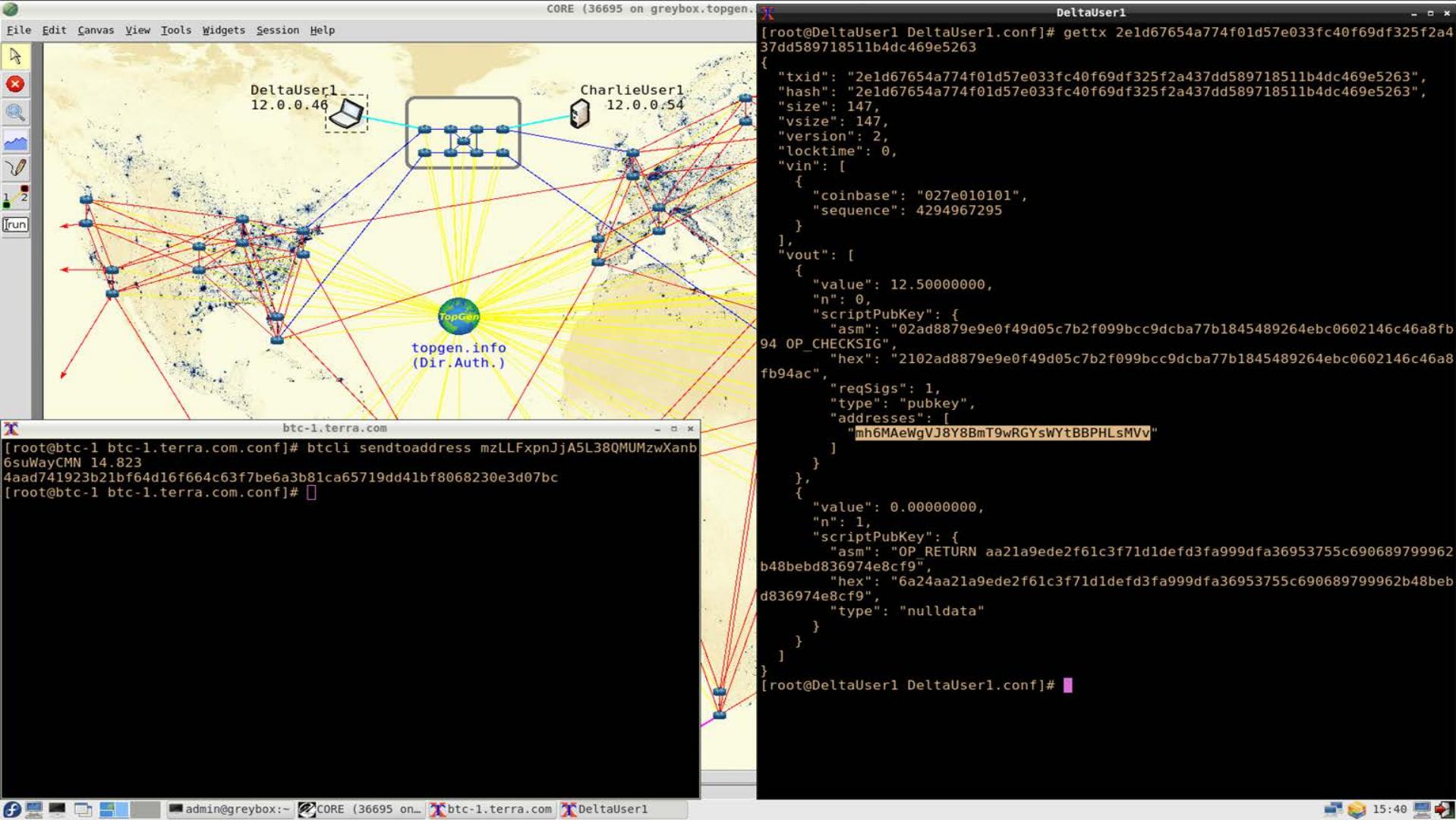


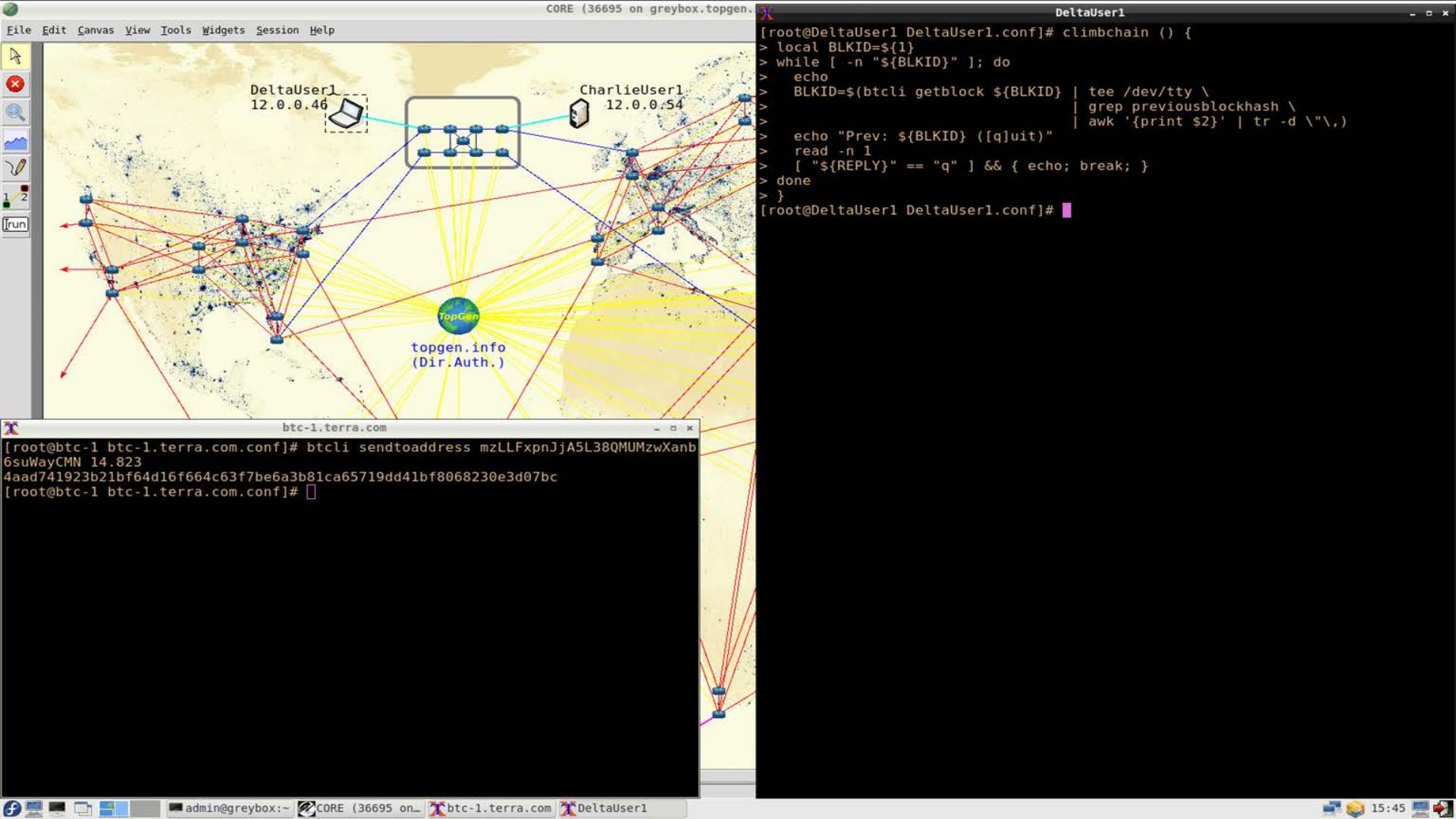


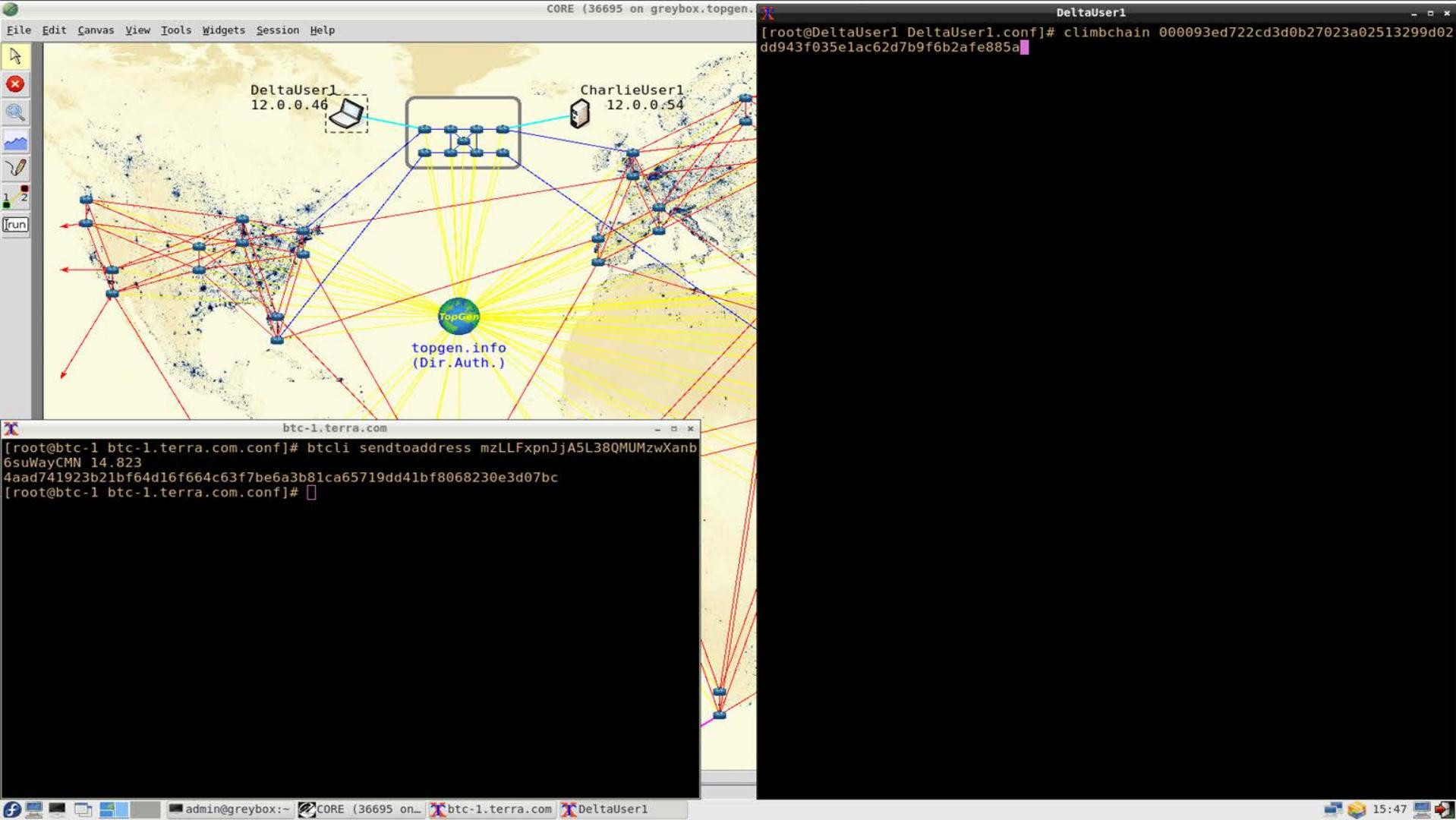
```
[root@btc-1 btc-1.terra.com.conf]# btcli sendtoaddress mzLLFxpnnJjA5L38QMUMzwan6suWayCn 14.823
4aad741923b21bf64d16f664c63f7be6a3b81ca65719dd41bf8068230e3d07bc
[root@btc-1 btc-1.terra.com.conf]#
```

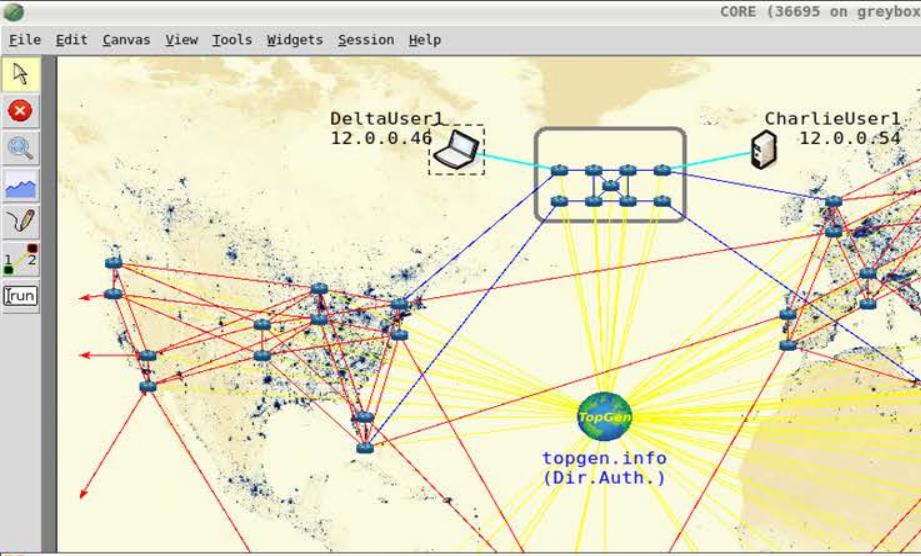




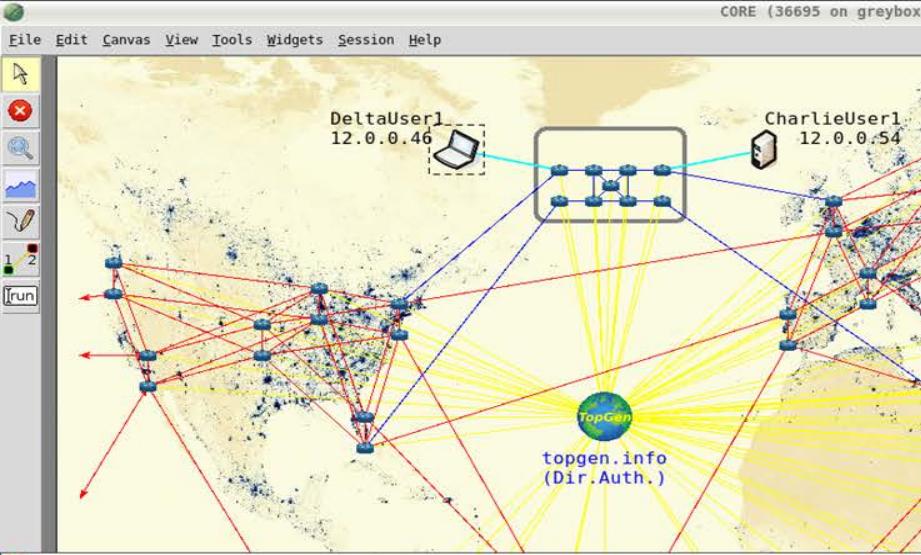




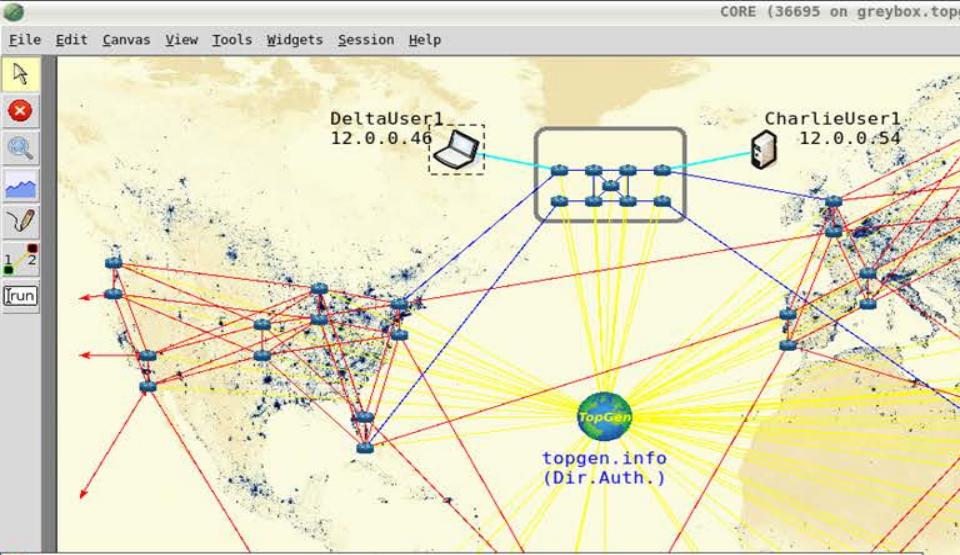




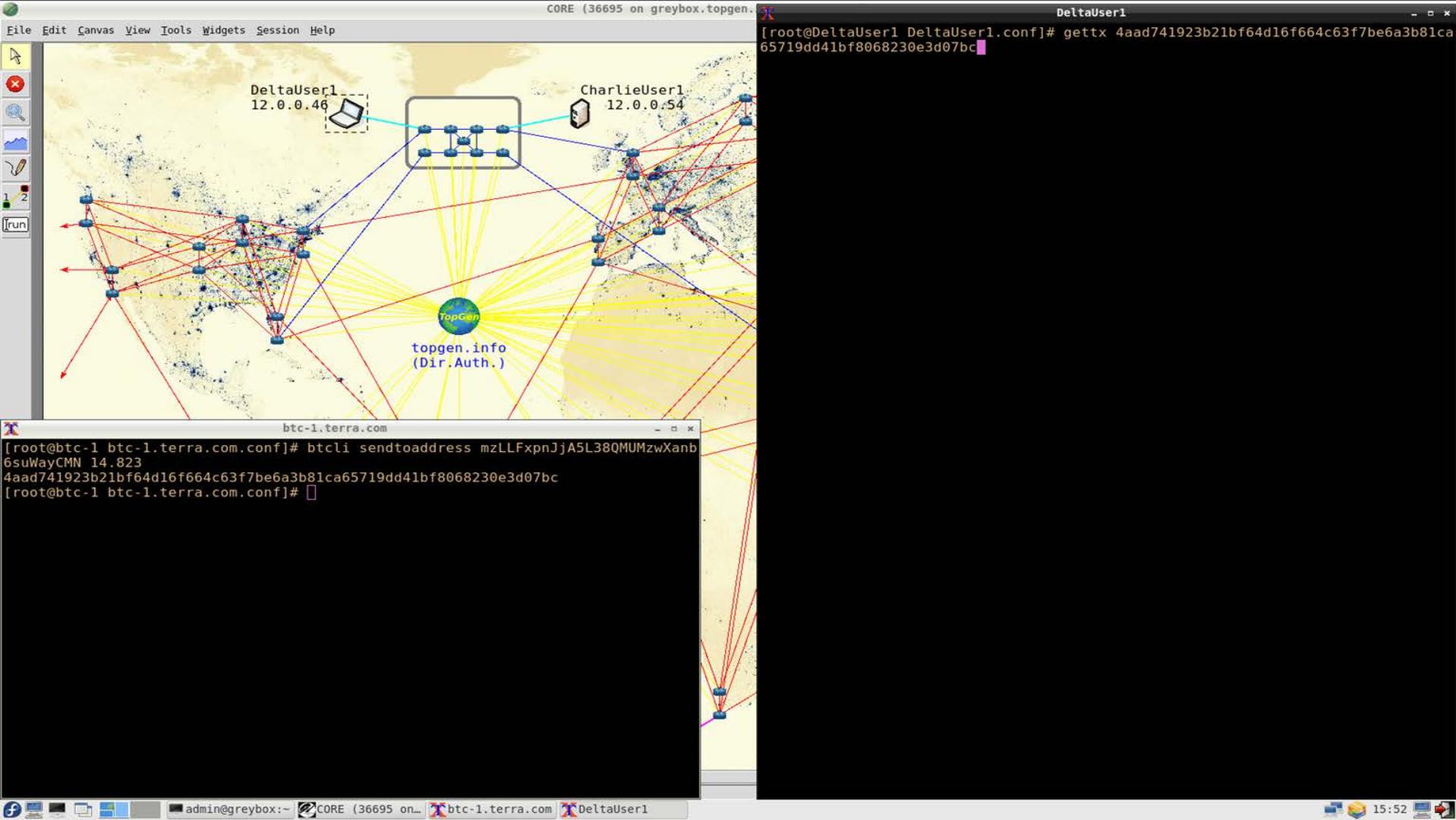
```
[root@btc-1 btc-1.terra.com.conf]# btcli sendtoaddress mzLLFxpnnJjA5L38QMUMzwXanb6suWayCMN 14.823
4aad741923b21bf64d16f664c63f7be6a3b81ca65719dd41bf8068230e3d07bc
[root@btc-1 btc-1.terra.com.conf]#
```

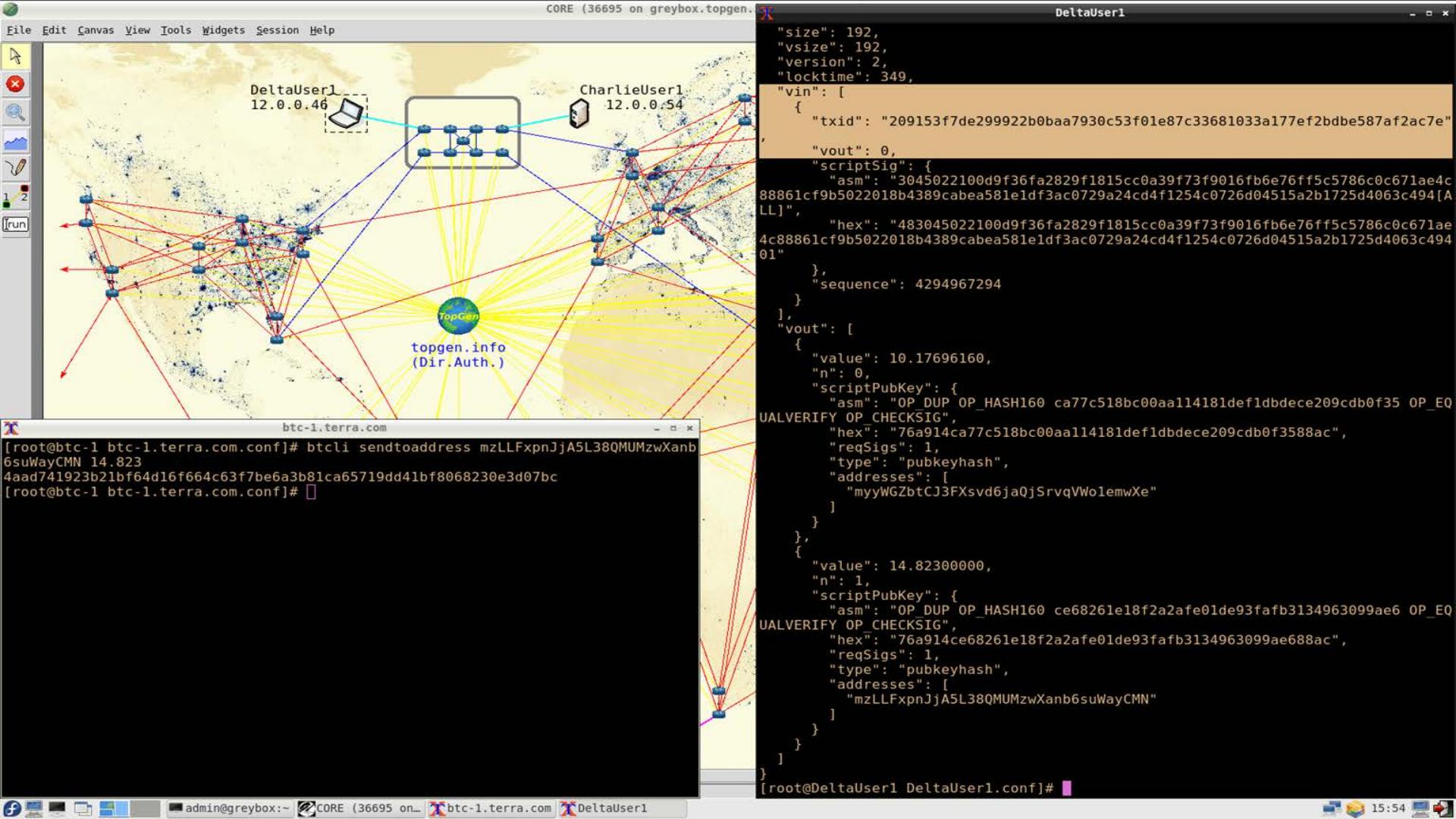


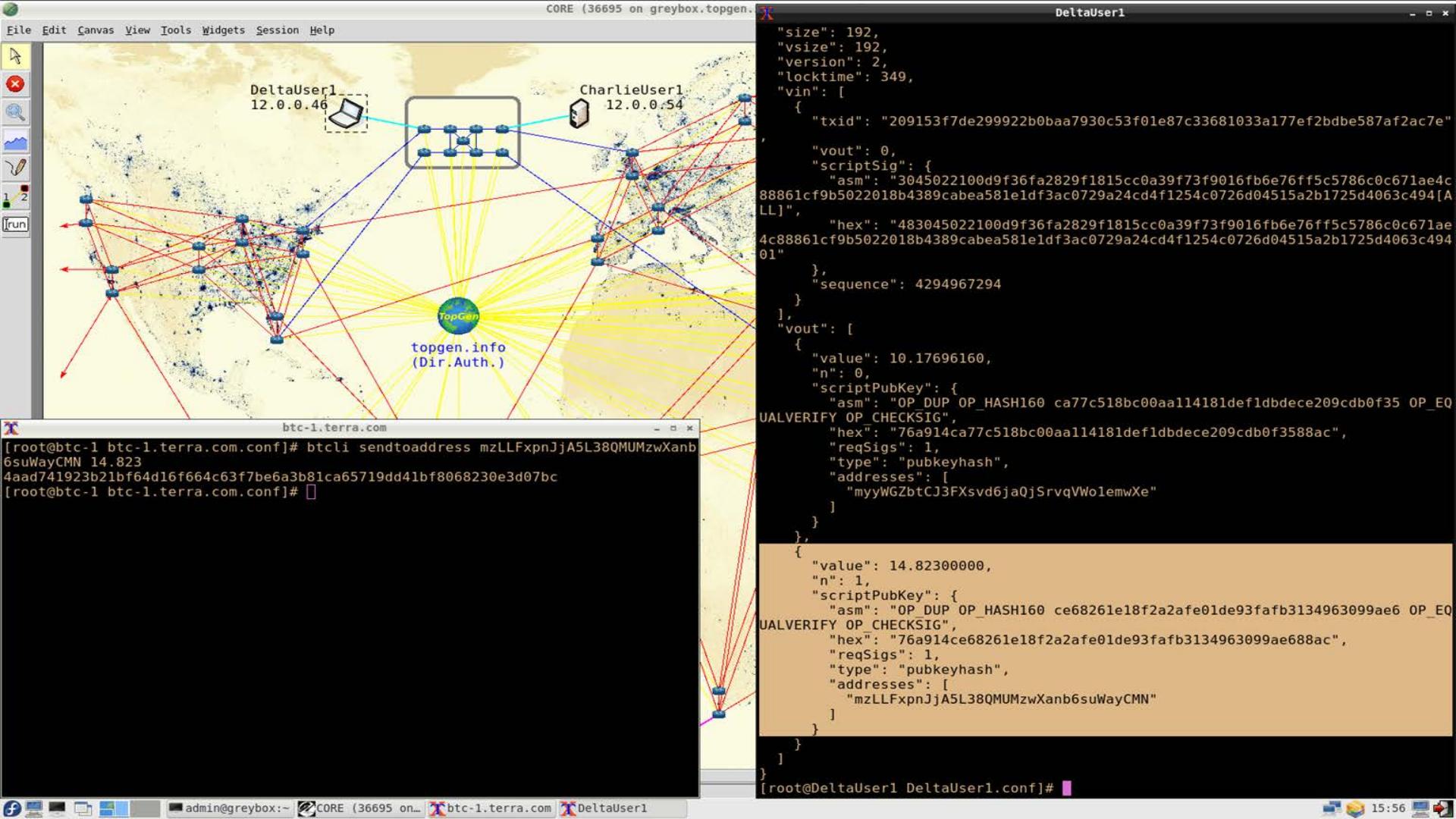
```
[root@btc-1 btc-1.terra.com.conf]# btcli sendtoaddress mzLLFxpnnJjA5L38QMUMzwXanb
6suWayCMN 14.823
4aad741923b21bf64d16f664c63f7be6a3b81ca65719dd41bf8068230e3d07bc
[root@btc-1 btc-1.terra.com.conf]#
```

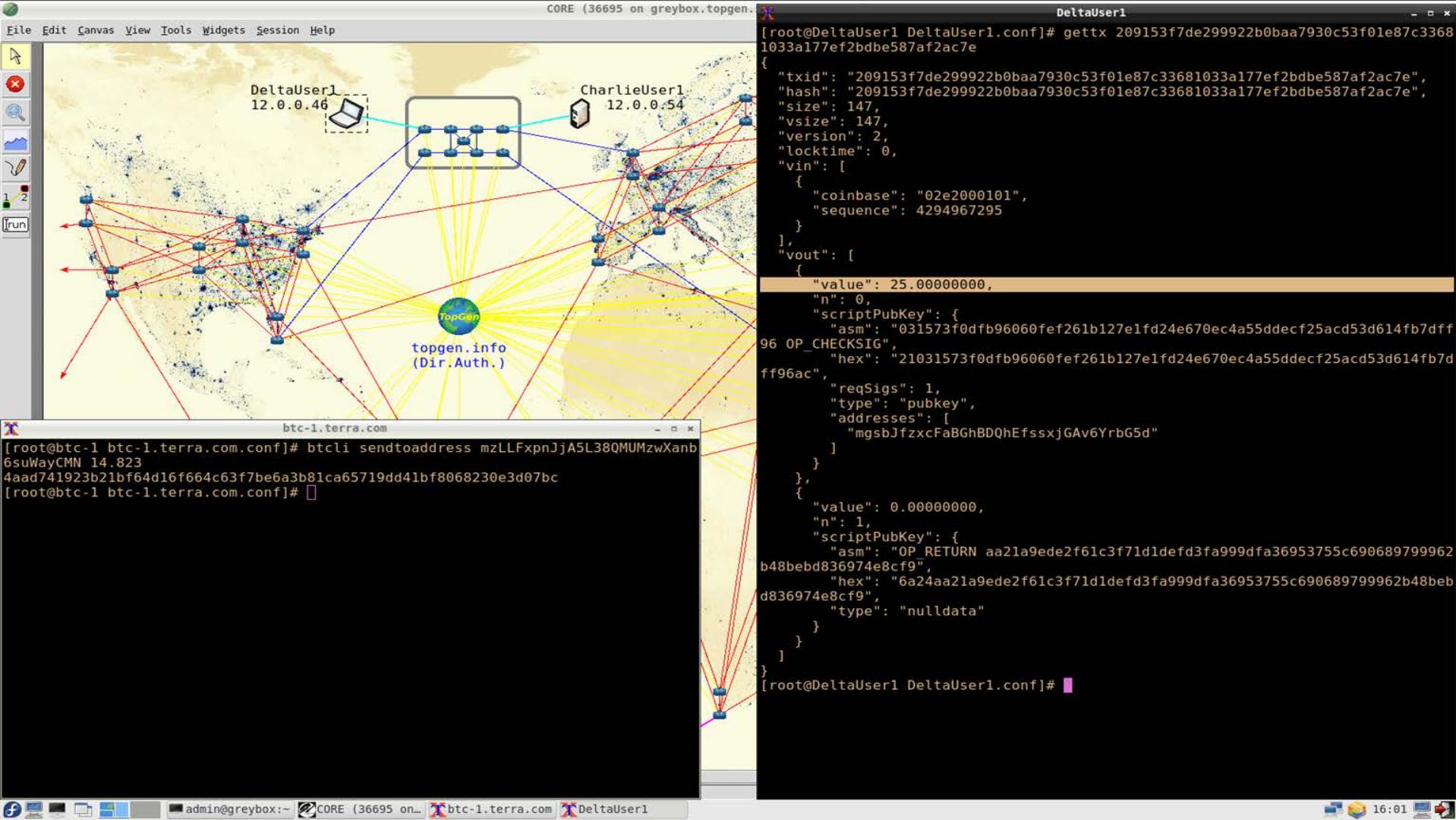


```
[root@btc-1 btc-1.terra.com.conf]# btcli sendtoaddress mzLLFxpnnJjA5L38QMUMzwXanb
6suWayCMN 14.823
4aad741923b21bf64d16f664c63f7be6a3b81ca65719dd41bf8068230e3d07bc
[root@btc-1 btc-1.terra.com.conf]#
```



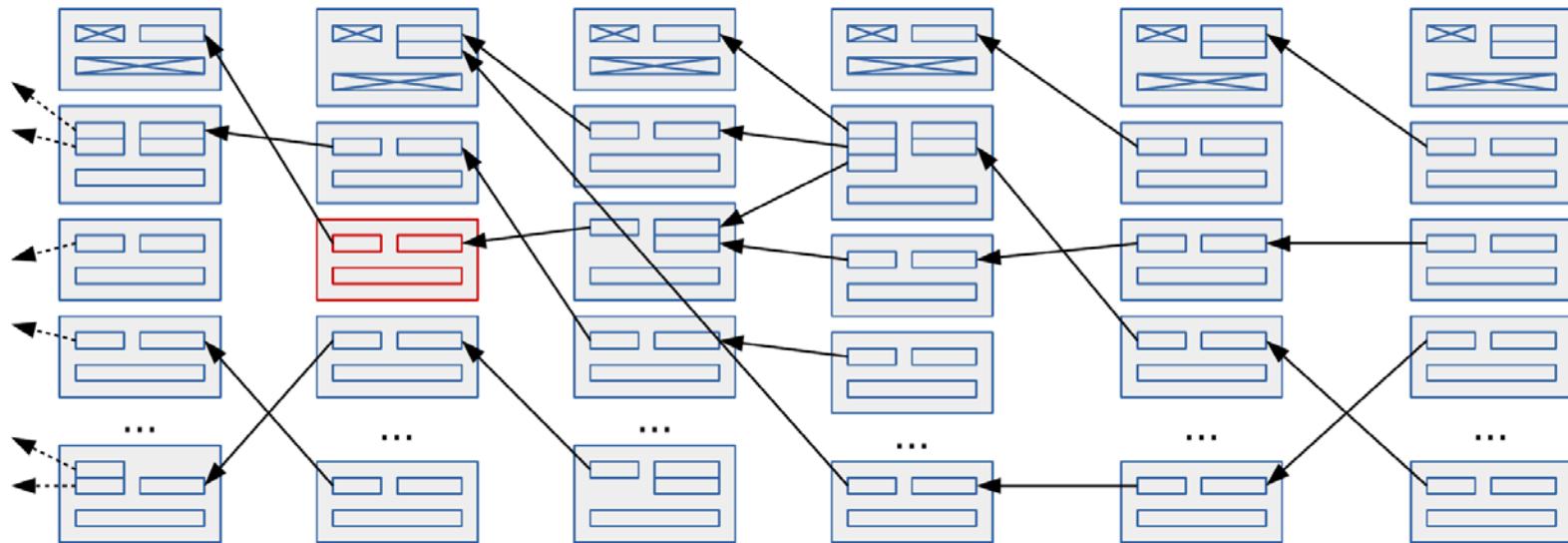




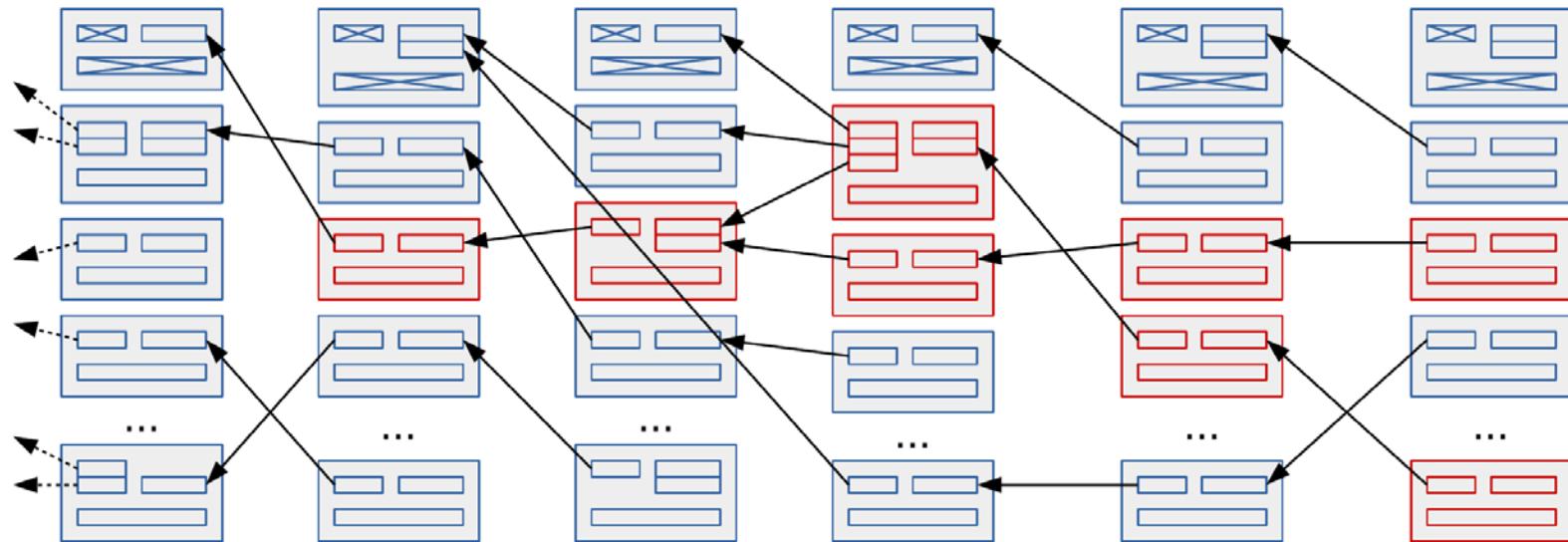




Tracing Transaction History



Tracing Transaction History



Blockchain programming is hard!

- Over **\$40M** were stolen from TheDAO due to a bug in the implementation (June 2016)
- **\$32M** were stolen due to a bug in a commonly used contract (June 2017)
- Bugs in smart contracts cannot be fixed after deployment

We want to build correct software, but current approaches have been shown to have security vulnerabilities

Obsidian: a new programming language

- Obsidian is a blockchain-based language with the goal of minimizing the risk of common security vulnerabilities
- Obsidian contains core features to allow users to write safe programs easily and effectively
- Obsidian programs consist of **contracts**, which contain fields, states, and **transactions**

Obsidian: a new programming language

Goals

- Make certain vulnerabilities impossible
- Make it easier to write correct programs
- Show effectiveness and correctness

Components

1. Typestate-oriented programming
2. Resource types

Typestate

- Blockchain programs commonly state-oriented
- Obsidian makes state first-class
 - An object in Obsidian has a **state** that restricts which **transactions** can be invoked on it.
- **State transitions** can change the state of an object

Typestate

```
contract LibraryPatron {  
    state NoCard {  
        transaction getCard() {  
            ...  
            ->HasCard;  
        }  
    }  
  
    state HasCard {  
        transaction getBook() {  
            ...  
        }  
    }  
}
```

- A LibraryPatron is always in either the NoCard or HasCard state

Typestate

```
contract LibraryPatron {  
    state NoCard {  
        transaction getCard() {  
            ...  
            ->HasCard;  
        }  
    }  
  
    state HasCard {  
        transaction getBook() {  
            ...  
        }  
    }  
}
```

- A `LibraryPatron` is always in either the `NoCard` or `HasCard` state
- `getBook` can only be called in `HasCard`; calling from `NoCard` state results in compile-time error

Typestate

```
contract LibraryPatron {  
    state NoCard {  
        transaction getCard() {  
            ...  
            ->HasCard;  
        }  
    }  
  
    state HasCard {  
        transaction getBook() {  
            ...  
        }  
    }  
}
```

- A `LibraryPatron` is always in either the `NoCard` or `HasCard` state.
- `getBook` can only be called in `HasCard`; calling from `NoCard` state results in compile-time error
- `->HasCard` is a state transition

Typestate – Other common applications

Voting

- Not Eligible
- Eligible, not voted
- Eligible, voted

Typestate – Other common applications

Supply chain

- Browsing
- Purchasing
- Order in processing
- Shipping
- Delivering to customer
- Return requested
- Delivering to business
- Returned

Typestate – Other common applications

Supply chain

- Browsing
- Purchasing
- Order in processing
- Shipping
- Delivering to customer
- Return requested
- Delivering to business
- Returned

The diagram consists of a vertical list of supply chain steps. Three red arrows point from the text "Customer management", "Manufacturing", and "Add steps for wholesale distributor" to the second, third, and fifth steps of the list, respectively.

- Browsing
- Purchasing
- Order in processing
- Shipping
- Delivering to customer
- Return requested
- Delivering to business
- Returned

Customer management

Manufacturing

Add steps for wholesale distributor

Linear Types

- Blockchain programs often manage some kind of resource
 - e.g., cryptocurrency, votes, items in supply chain
- **Linear types** allow the compiler to enforce “resource safety”:
 - Resources cannot be used more than once
 - Resources must be used before leaving the current scope (i.e., don’t lose it)

Linear Types

```
resource contract Money {...};  
  
contract Account {  
    Money balance;  
  
    transaction closeAccount(Account a) {  
        a.withdraw(balance);  
        balance = new Money(0);  
    }  
  
    transaction withdraw(Money m) {...}  
}
```

- Financial application example
- balance is a type of Money, which is a *linear type*

Linear Types

```
resource contract Money {...};

contract Account {
    Money balance;

    transaction closeAccount(Account a) {
        a.withdraw(balance);
        balance = new Money(0);
    }

    transaction withdraw(Money m) {...}
}
```

- When `balance` is moved out of scope, we need to create a new `Money` object to replace it
- The new `Money` is created with a value of 0
- Note that we're not referring to the actual amount of money, we're referring to the code used to track the money
- Code security, not accounting

Linear Types

```
resource contract Money {...};

contract Account {
    Money balance;

    transaction closeAccount(Account a) {
        a.withdraw(balance);
        a.withdraw(balance);
        balance = new Money(0);
    }

    transaction withdraw(Money m) {...}
}
```

- Introducing bug... spending more Money than available
- Program would fail when trying to compile, rather than when the user tried to run the program

Usability

Programmers should be able to write correct Obsidian code easily and effectively.
Creating an intuitive language is hard! Many difficult design choices exist

Usability

```
contract C {
    state S1 {
        transaction t(int x) {
            ->S2(x);
        }
    }

    state S2 {
        int x1;
        S2(int x) {
            x1 = x;
            ->S3(x1);
        }
    }

    state S3 {
        ...
    }
}
```

```
contract C {
    state S1 {
        transaction t(int x) {
            ->S2{x1 = x};
            toS3();
        }
    }

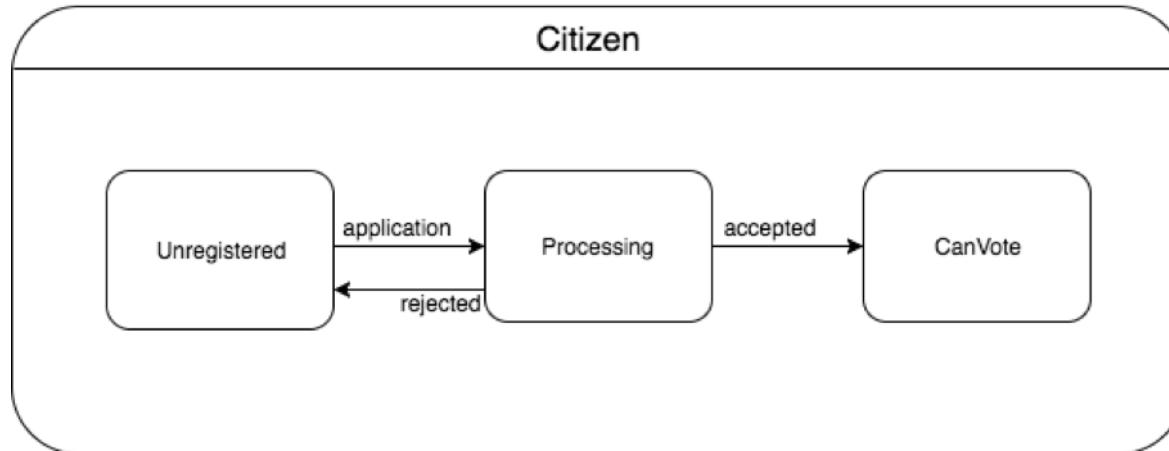
    state S2 {
        int x1;
        transaction toS3() {
            ->S3{x2 = x1};
        }
    }

    state S3{
        ...
    }
}
```

Which is “correct”?

Usability study

Participants were given a description of a voter registration system for a hypothetical democratic nation.



Usability study

1. Write pseudocode to implement program.
2. Given a state diagram modeling the voter registration system, modify pseudocode.
3. Given Obsidian tutorial (with no information on state transitions) invent syntax for state transitions and complete an Obsidian contract.
4. Shown three options for state transitions, complete a brief contract for each option.
5. Choose one of the three options and use it to complete the Obsidian program from part 3.

Usability study – Findings

- Programmers do not naturally consider state-based design when architecting code
- Most intuitive design: include all possible state actions explicitly within the state

Summary

Bitcoin simulator

- Virtual implementation of Bitcoin network
- Useful for forensic analysis

Obsidian

- Secure-by-design language for blockchain development
- Typestate and linear resources help users write safe programs easily and effectively
- Usable programming language design requires iteration and user testing

Contact Information

Presenters

Elli Kanal

Technical Manager

Email: ekanal@cert.org

Gabe Somlo

Cybersecurity Researcher

Email: glsomlo@cert.org

