From Technofeudalism to Technosocialism: FORTH unexpected roles in Decentralised Systems

https://www.linkedin.com/in/liang-ng/

https://github.com/omnixtar/omnixtar.github.io/blob/main/docs/FORTH_Decentralised_Systems_2024.pdf

Part A: Introduction

We began this presentation with a brief description of a FORTH like engine in Bitcoin (Part B) and how the invisible internet project (Part B), being a crucial alternative to the domain name system, was used as the backbone of Bitcoin but remains relatively obscure.

We then extended the concept of Bitcoin address, as the hash of public key, to Omnihash user identifier and Omnihash JSON, being a generalised representation of digital assets with Decentralised ownerships. (Part C)

We further propose to employ a FORTH like engine in shell libraries for JavaScript and Java applications (Part D: Trojan Horse Super-Shell), in order to consolidate fragmented user base of open source projects and small commercial social media applications to compete with and create one to one replacements for mainstream dominant social media platforms (Goal of Decentralised Systems).

From Technofeudalism to Technosocialism: FORTH unexpected roles in Decentralised Systems

Technofeudalism is a term coined by former Greek finance minister Yanis Varoufakis as the title of a book which compares top American technology companies to feudal lords, who literally enslaved individual users and free software programmers, exploiting their data and source code without fairly rewarding them.

Technosocialism has been used by many other authors but I use it to describe the very real but underreported progress made by free software programmers and free software, despite the lack of fair mechanisms for rewards so far. Free software programmers or technosocialists have made technofeudalism possible as a matter of fact, but they do not get to enjoy the rewards.

Bitcoin and related projects were supposed to address this problem, and are successful in their own way, so we are now half way towards technosocialism.

Part B:
FORTH-like Engine in Bitcoin
+ Invisible Internet Project

Script

Main page
Bitcoin FAQ
Editing help
Forums
Chatrooms
Recent changes
Page index

Tools

What links here
Related changes
Special pages
Printable version
Permanent link
Page information

Sister projects

Bitcoin uses a scripting system for transactions. Forth-like, **Script** is simple, stack-based, and processed from left to right. It is intentionally not Turing-complete, with no loops.

A script is essentially a list of instructions recorded with each transaction that describe how the next person wanting to spend the Bitcoins being transferred can gain access to them. The script for a typical Bitcoin transfer to destination Bitcoin address D simply encumbers future spending of the bitcoins with two things: the spender must provide

1. a public key that, when hashed, yields destination address D embedded in the script, and
2. a signature to prove ownership of the private key corresponding to the public key just provided.

Scripting provides the flexibility to change the parameters of what's needed to spend transferred Bitcoins. For example, the scripting system could be used to require two private keys, or a combination of several keys, or even no keys at all.

A transaction is valid if nothing in the combined script triggers failure and the top stack item is True (non-zero) when the script exits. The party that originally *sent* the Bitcoins now being spent

OP_FROMALTSTACK	108	0x6c	(alt)x1	x1	Puts the input onto the top of the main stack. Removes it from the alt stack.
OP_IFDUP	115	0x73	x	x / x x	If the top stack value is not 0, duplicate it.
OP_DEPTH	116	0x74	Nothing	<Stack size>	Puts the number of stack items onto the stack.
OP_DROP	117	0x75	x	Nothing	Removes the top stack item.
OP_DUP	118	0x76	x	x x	Duplicates the top stack item.
OP_NIP	119	0x77	x1 x2	x2	Removes the second-to-top stack item.
OP_OVER	120	0x78	x1 x2	x1 x2 x1	Copies the second-to-top stack item to the top.
OP_PICK	121	0x79	xn ... x2 x1 x0 <n>	xn ... x2 x1 x0 xn	The item *n* back in the stack is copied to the top.
OP_ROLL	122	0x7a	xn ... x2 x1 x0 <n>	... x2 x1 x0 xn	The item *n* back in the stack is moved to the top.
OP_ROT	123	0x7b	x1 x2 x3	x2 x3 x1	The 3rd item down the stack is moved to the top.

OP_NOP10	179-185	0xb3-0xb9	transaction as invalid.

Script examples

The following is a list of interesting scripts. When notating scripts, data to be pushed to the stack is generally enclosed in angle brackets and data push commands are omitted. Non-bracketed words are opcodes. These examples include the “OP_” prefix, but it is permissible to omit it. Thus “<pubkey1> <pubkey2> OP_2 OP_CHECKMULTISIG” may be abbreviated to “<pubkey1> <pubkey2> 2 CHECKMULTISIG”. Note that there is a small number of standard script forms that are relayed from node to node; non-standard scripts are accepted if they are in a block, but nodes will not relay them.

Standard Transaction to Bitcoin address (pay-to-pubkey-hash)

```
scriptPubKey: OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
scriptSig: <sig> <pubKey>
```

To demonstrate how scripts look on the wire, here is a raw scriptPubKey:

```
  76       A9             14
OP_DUP OP_HASH160    Bytes to push

89 AB CD EF AB BA AB BA AB BA AB BA AB BA AB BA AB BA   88         AC
                     Data to push                     OP_EQUALVERIFY
```

Stack	Script	Description
Empty.	<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	scriptSig and scriptPubKey are combined.
<sig> <pubKey>	OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	Constants are added to the stack.
<sig> <pubKey> <pubKey>	OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	Top stack item is duplicated.
<sig> <pubKey> <pubHashA>	<pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	Top stack item is hashed.
<sig> <pubKey> <pubHashA> <pubKeyHash>	OP_EQUALVERIFY OP_CHECKSIG	Constant added.
<sig> <pubKey>	OP_CHECKSIG	Equality is checked between the top two stack items.
true	Empty.	Signature is checked for top two stack items.

Obsolete pay-to-pubkey transaction

https://github.com/bitcoin/bitcoin/blob/master/src/script/interpreter.cpp

Code Blame

```
406  bool EvalScript(std::vector<std::vector<unsigned char> >& stack, const CScript& script, unsigned int fla
407  {
408      static const CScriptNum bnZero(0);
409      static const CScriptNum bnOne(1);
410      // static const CScriptNum bnFalse(0);
411      // static const CScriptNum bnTrue(1);
412      static const valtype vchFalse(0);
413      // static const valtype vchZero(0);
414      static const valtype vchTrue(1, 1);
415
416      // sigversion cannot be TAPROOT here, as it admits no script execution.
417      assert(sigversion == SigVersion::BASE || sigversion == SigVersion::WITNESS_V0 || sigversion == SigV
418
419      CScript::const_iterator pc = script.begin();
420      CScript::const_iterator pend = script.end();
421      CScript::const_iterator pbegincodehash = script.begin();
422      opcodetype opcode;
423      valtype vchPushValue;
424      ConditionStack vfExec;
```

```
406  bool EvalScript(std::vector<std::vector<unsigned char> >& stack, const CScript& script, unsigned int fl
435
436      try
437      {
438          for (; pc < pend; ++opcode_pos) {
439              bool fExec = vfExec.all_true();
440
441              //
442              // Read instruction
443              //
444              if (!script.GetOp(pc, opcode, vchPushValue))
445                  return set_error(serror, SCRIPT_ERR_BAD_OPCODE);
446              if (vchPushValue.size() > MAX_SCRIPT_ELEMENT_SIZE)
447                  return set_error(serror, SCRIPT_ERR_PUSH_SIZE);
448
449              if (sigversion == SigVersion::BASE || sigversion == SigVersion::WITNESS_V0) {
450                  // Note how OP_RESERVED does not count towards the opcode limit.
451                  if (opcode > OP_16 && ++nOpCount > MAX_OPS_PER_SCRIPT) {
452                      return set_error(serror, SCRIPT_ERR_OP_COUNT);
453                  }
```

```
406     bool EvalScript(std::vector<std::vector<unsigned char> >& stack, const CScript& script, unsigned int fl

    if (opcode == OP_CAT ||
        opcode == OP_SUBSTR ||
        opcode == OP_LEFT ||
        opcode == OP_RIGHT ||
        opcode == OP_INVERT ||
        opcode == OP_AND ||
        opcode == OP_OR ||
        opcode == OP_XOR ||
        opcode == OP_2MUL ||
        opcode == OP_2DIV ||
        opcode == OP_MUL ||
        opcode == OP_DIV ||
        opcode == OP_MOD ||
        opcode == OP_LSHIFT ||
        opcode == OP_RSHIFT)
        return set_error(serror, SCRIPT_ERR_DISABLED_OPCODE); // Disabled opcodes (CVE-2010-5137).

    // With SCRIPT_VERIFY_CONST_SCRIPTCODE, OP_CODESEPARATOR in non-segwit script is rejected even in an unexecu
```

```
406   bool EvalScript(std::vector<std::vector<unsigned char> >& stack, const CScript& script, unsigned int fl
483               switch (opcode)
484               {
485                   //
486                   // Push value
487                   //
488                   case OP_1NEGATE:
489                   case OP_1:
490                   case OP_2:
491                   case OP_3:
492                   case OP_4:
493                   case OP_5:
494                   case OP_6:
495                   case OP_7:
496                   case OP_8:
497                   case OP_9:
498                   case OP_10:
499                   case OP_11:
500                   case OP_12:
```

```
406  bool EvalScript(std::vector<std::vector<unsigned char> >& stack, const CScript& script, unsigned int fl
515                  //
516                  // Control
517                  //
518                  case OP_NOP:
519                      break;
520
521                  case OP_CHECKLOCKTIMEVERIFY:
522                  {
523                      if (!(flags & SCRIPT_VERIFY_CHECKLOCKTIMEVERIFY)) {
524                          // not enabled; treat as a NOP2
525                          break;
526                      }
527
528                      if (stack.size() < 1)
529                          return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
530
531                      // Note that elsewhere numeric opcodes are limited to
532                      // operands in the range -2**31+1 to 2**31-1, however it is
```

```
406  bool EvalScript(std::vector<std::vector<unsigned char> >& stack, const CScript& script, unsigned int fl
601
602                  case OP_IF:
603                  case OP_NOTIF:
604                  {
605                      // <expression> if [statements] [else [statements]] endif
606                      bool fValue = false;
607                      if (fExec)
608                      {
609                          if (stack.size() < 1)
610                              return set_error(serror, SCRIPT_ERR_UNBALANCED_CONDITIONAL);
611                          valtype& vch = stacktop(-1);
612                          // Tapscript requires minimal IF/NOTIF inputs as a consensus rule.
613                          if (sigversion == SigVersion::TAPSCRIPT) {
614                              // The input argument to the OP_IF and OP_NOTIF opcodes must be either
615                              // exactly 0 (the empty vector) or exactly 1 (the one-byte vector with valu
616                              if (vch.size() > 1 || (vch.size() == 1 && vch[0] != 1)) {
617                                  return set_error(serror, SCRIPT_ERR_TAPSCRIPT_MINIMALIF);
618                              }
619                          }
```

```
406 bool EvalScript(std::vector<std::vector<unsigned char> >& stack, const CScript& script, unsigned int fl
635
636                     case OP_ELSE:
637                     {
638                         if (vfExec.empty())
639                             return set_error(serror, SCRIPT_ERR_UNBALANCED_CONDITIONAL);
640                         vfExec.toggle_top();
641                     }
642                     break;
643
644                     case OP_ENDIF:
645                     {
646                         if (vfExec.empty())
647                             return set_error(serror, SCRIPT_ERR_UNBALANCED_CONDITIONAL);
648                         vfExec.pop_back();
649                     }
650                     break;
651
652                     case OP_VERIFY:
653                     {
```

```
406     bool EvalScript(std::vector<std::vector<unsigned char> >& stack, const CScript& script, unsigned int fl
673                 //
674                 // Stack ops
675                 //
676                 case OP_TOALTSTACK:
677                 {
678                     if (stack.size() < 1)
679                         return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
680                     altstack.push_back(stacktop(-1));
681                     popstack(stack);
682                 }
683                 break;
684
685                 case OP_FROMALTSTACK:
686                 {
687                     if (altstack.size() < 1)
688                         return set_error(serror, SCRIPT_ERR_INVALID_ALTSTACK_OPERATION);
689                     stack.push_back(altstacktop(-1));
690                     popstack(altstack);
```

```
406  bool EvalScript(std::vector<std::vector<unsigned char> >& stack, const CScript& script, unsigned int fl
693
694                  case OP_2DROP:
695                  {
696                      // (x1 x2 -- )
697                      if (stack.size() < 2)
698                          return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
699                      popstack(stack);
700                      popstack(stack);
701                  }
702                  break;
703
704                  case OP_2DUP:
705                  {
706                      // (x1 x2 -- x1 x2 x1 x2)
707                      if (stack.size() < 2)
708                          return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
709                      valtype vch1 = stacktop(-2);
710                      valtype vch2 = stacktop(-1);
711                      stack.push_back(vch1);
```

```
406  bool EvalScript(std::vector<std::vector<unsigned char> >& stack, const CScript& script, unsigned int fl
703
704                  case OP_2DUP:
705                  {
706                      // (x1 x2 -- x1 x2 x1 x2)
707                      if (stack.size() < 2)
708                          return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
709                      valtype vch1 = stacktop(-2);
710                      valtype vch2 = stacktop(-1);
711                      stack.push_back(vch1);
712                      stack.push_back(vch2);
713                  }
714                  break;
715
716                  case OP_3DUP:
717                  {
718                      // (x1 x2 x3 -- x1 x2 x3 x1 x2 x3)
719                      if (stack.size() < 3)
720                          return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
721                      valtype vch1 = stacktop(-3);
```

```
406     bool EvalScript(std::vector<std::vector<unsigned char> >& stack, const CScript& script, unsigned int fl
730                 case OP_2OVER:
731                 {
732                     // (x1 x2 x3 x4 -- x1 x2 x3 x4 x1 x2)
733                     if (stack.size() < 4)
734                         return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
735                     valtype vch1 = stacktop(-4);
736                     valtype vch2 = stacktop(-3);
737                     stack.push_back(vch1);
738                     stack.push_back(vch2);
739                 }
740                 break;
741
742                 case OP_2ROT:
743                 {
744                     // (x1 x2 x3 x4 x5 x6 -- x3 x4 x5 x6 x1 x2)
745                     if (stack.size() < 6)
746                         return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
747                     valtype vch1 = stacktop(-6);
```

```
406     bool EvalScript(std::vector<std::vector<unsigned char> >& stack, const CScript& script, unsigned int fl

755                 case OP_2SWAP:
756                 {
757                     // (x1 x2 x3 x4 -- x3 x4 x1 x2)
758                     if (stack.size() < 4)
759                         return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
760                     swap(stacktop(-4), stacktop(-2));
761                     swap(stacktop(-3), stacktop(-1));
762                 }
763                 break;
764
765                 case OP_IFDUP:
766                 {
767                     // (x - 0 | x x)
768                     if (stack.size() < 1)
769                         return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
770                     valtype vch = stacktop(-1);
771                     if (CastToBool(vch))
772                         stack.push_back(vch);
```

```
406  bool EvalScript(std::vector<std::vector<unsigned char> >& stack, const CScript& script, unsigned int fl
765                  case OP_IFDUP:
766                  {
767                      // (x - 0 | x x)
768                      if (stack.size() < 1)
769                          return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
770                      valtype vch = stacktop(-1);
771                      if (CastToBool(vch))
772                          stack.push_back(vch);
773                  }
774                  break;
775
776                  case OP_DEPTH:
777                  {
778                      // -- stacksize
779                      CScriptNum bn(stack.size());
780                      stack.push_back(bn.getvch());
781                  }
782                  break;
```

```
406  bool EvalScript(std::vector<std::vector<unsigned char> >& stack, const CScript& script, unsigned int fl
783
784                  case OP_DROP:
785                  {
786                      // (x -- )
787                      if (stack.size() < 1)
788                          return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
789                      popstack(stack);
790                  }
791                  break;
792
793                  case OP_DUP:
794                  {
795                      // (x -- x x)
796                      if (stack.size() < 1)
797                          return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
798                      valtype vch = stacktop(-1);
799                      stack.push_back(vch);
800                  }
801                  break;
```

```
406  bool EvalScript(std::vector<std::vector<unsigned char> >& stack, const CScript& script, unsigned int fl
803                  case OP_NIP:
804                  {
805                      // (x1 x2 -- x2)
806                      if (stack.size() < 2)
807                          return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
808                      stack.erase(stack.end() - 2);
809                  }
810                  break;
811
812                  case OP_OVER:
813                  {
814                      // (x1 x2 -- x1 x2 x1)
815                      if (stack.size() < 2)
816                          return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
817                      valtype vch = stacktop(-2);
818                      stack.push_back(vch);
819                  }
820                  break;
```

```
406  bool EvalScript(std::vector<std::vector<unsigned char> >& stack, const CScript& script, unsigned int fl
822                  case OP_PICK:
823                  case OP_ROLL:
824                  {
825                      // (xn ... x2 x1 x0 n - xn ... x2 x1 x0 xn)
826                      // (xn ... x2 x1 x0 n - ... x2 x1 x0 xn)
827                      if (stack.size() < 2)
828                          return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
829                      int n = CScriptNum(stacktop(-1), fRequireMinimal).getint();
830                      popstack(stack);
831                      if (n < 0 || n >= (int)stack.size())
832                          return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
833                      valtype vch = stacktop(-n-1);
834                      if (opcode == OP_ROLL)
835                          stack.erase(stack.end()-n-1);
836                      stack.push_back(vch);
837                  }
838                  break;
839
```

```
406  bool EvalScript(std::vector<std::vector<unsigned char> >& stack, const CScript& script, unsigned int fl
840                  case OP_ROT:
841                  {
842                      // (x1 x2 x3 -- x2 x3 x1)
843                      //  x2 x1 x3  after first swap
844                      //  x2 x3 x1  after second swap
845                      if (stack.size() < 3)
846                          return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
847                      swap(stacktop(-3), stacktop(-2));
848                      swap(stacktop(-2), stacktop(-1));
849                  }
850                  break;
851
852                  case OP_SWAP:
853                  {
854                      // (x1 x2 -- x2 x1)
855                      if (stack.size() < 2)
856                          return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
857                      swap(stacktop(-2), stacktop(-1));
858                  }
```

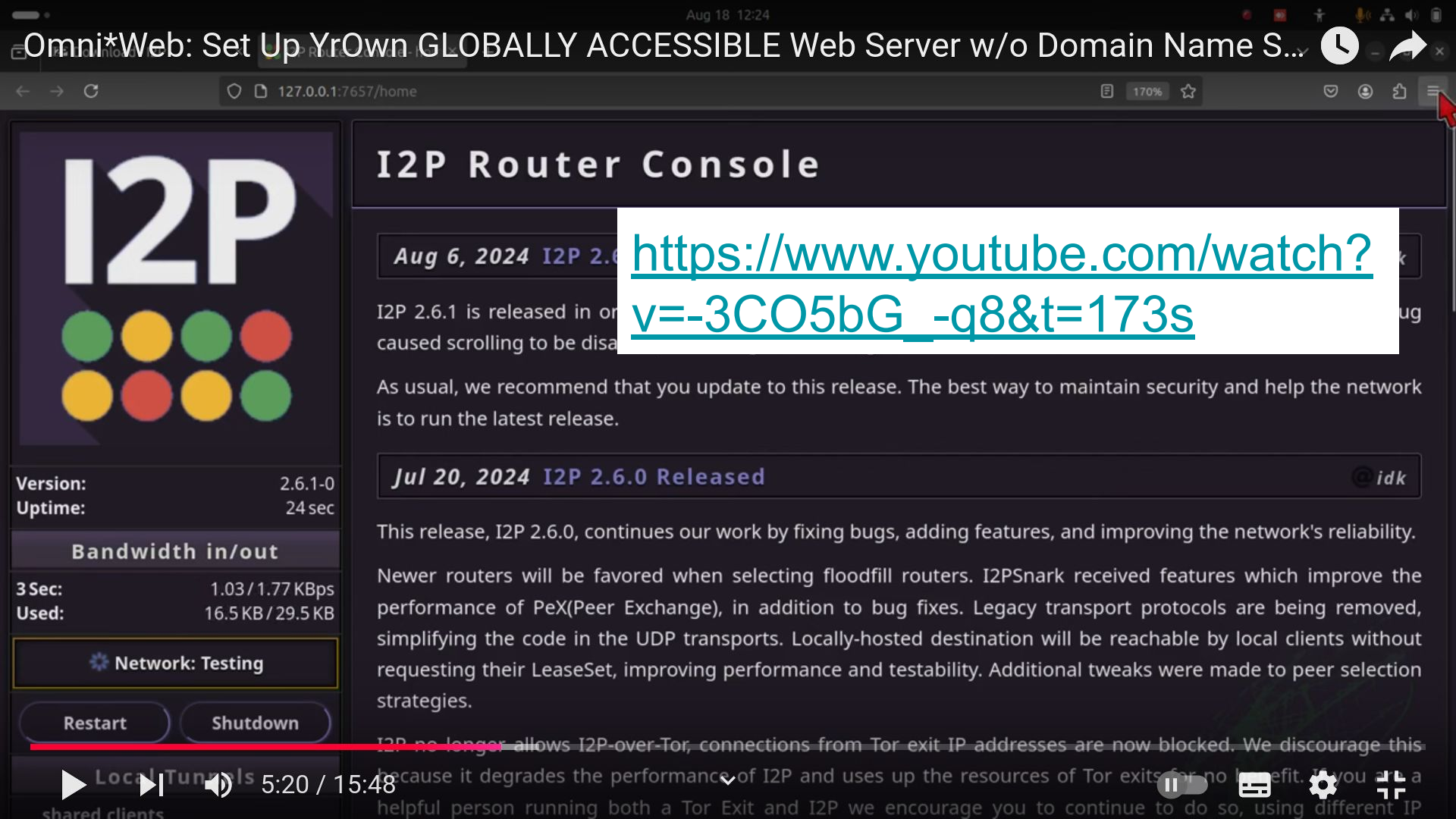
Part B: Invisible Internet Project

- Potential, size, replace DNS, sharing costs become revenue

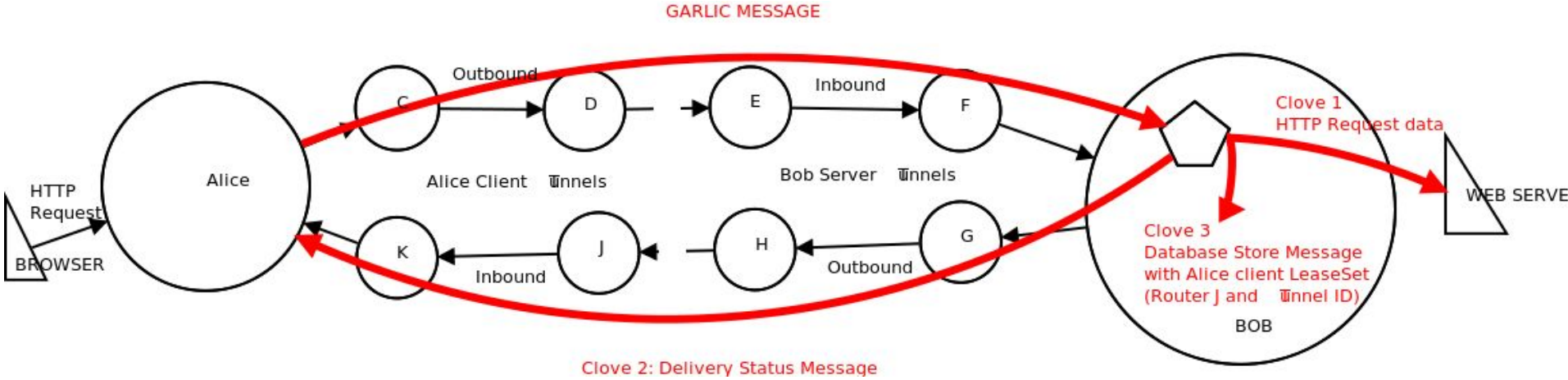
The **invisible internet project (I2P)** is actually *a replacement of the domain name system and the backbone of bitcoin.* It has the potential to *enable every home user to become a cloud service provider* by *sharing the processing power, network bandwidth and storage*, **turning costs of renting cloud services into revenues!!**

Unfortunately, the core developer team did not pursue further beyond *focusing on privacy aspects*, leading to our suspicion that they might have made billions of dollars by being *the earliest contributors of bitcoin*.

Paranoid of “Darknet” – but *I2P router in Java* feels safer – has *FORTH-shell in Java (Phoscript engine)* – effectively a *debugger inside I2P – or Bitcoin* – think about its potential – *“Trojan horse super-shell”* in Part D.


Omni*Web: Set Up YrOwn GLOBALLY ACCESSIBLE Web Server w/o Domain Name S...
127.0.0.1:7657/home
I2P Router Console
https://www.youtube.com/watch?v=-3CO5bG_-q8&t=173s
Aug 6, 2024 I2P 2.6
I2P 2.6.1 is released in or
caused scrolling to be disa
As usual, we recommend that you update to this release. The best way to maintain security and help the network is to run the latest release.
Jul 20, 2024 I2P 2.6.0 Released
idk
This release, I2P 2.6.0, continues our work by fixing bugs, adding features, and improving the network's reliability.
Newer routers will be favored when selecting floodfill routers. I2PSnark received features which improve the performance of PeX(Peer Exchange), in addition to bug fixes. Legacy transport protocols are being removed, simplifying the code in the UDP transports. Locally-hosted destination will be reachable by local clients without requesting their LeaseSet, improving performance and testability. Additional tweaks were made to peer selection strategies.
I2P no longer allows I2P-over-Tor, connections from Tor exit IP addresses are now blocked. We discourage this because it degrades the performance of I2P and uses up the resources of Tor exits for no benefit. If you are a helpful person running both a Tor Exit and I2P we encourage you to continue to do so, using different IP
Version: 2.6.1-0
Uptime: 24 sec
Bandwidth in/out
3 Sec: 1.03 / 1.77 KBps
Used: 16.5 KB / 29.5 KB
Network: Testing
Restart
Shutdown
Local Tunnels
shared clients
5:20 / 15:48

https://geti2p.net/en/docs/how/garlic-routing



End-to-End Message Bundling

At the layer above tunnels, I2P delivers end-to-end messages between Destinations. Just as within a single tunnel, we use ElGamal/AES+SessionTag for the encryption. Each client message as delivered to the router through the I2CP interface becomes a single Garlic Clove with its own Delivery Instructions, inside a Garlic Message. Delivery Instructions may specify a Destination, Router, or Tunnel.

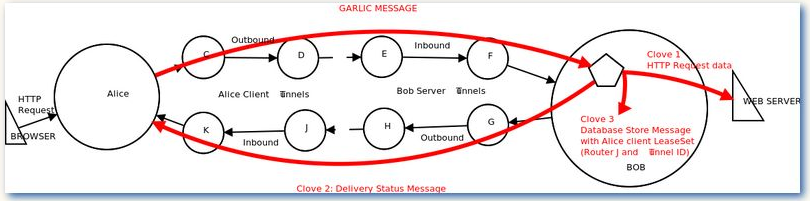
Generally, a Garlic Message will contain only one clove. However, the router will periodically bundle two additional cloves in the Garlic Message

A Delivery Status Message, with Delivery Instructions specifying that it be sent back to the originating router as an acknowledgment. This is similar to the "reply block" or "reply onion" described in the references. It is used for determining the success or failure of end to end message delivery. The originating router may, upon failure to receive the Delivery Status Message within the

End-to-End Message Bundling

At the layer above tunnels, I2P delivers end-to-end messages between Destinations. Just as within a single tunnel, we use ElGamal/AES+SessionTag for the encryption. Each client message as delivered to the router through the I2CP interface becomes a single Garlic Clove with its own Delivery Instructions, inside a Garlic Message. Delivery Instructions may specify a Destination, Router, or Tunnel.

Generally, a Garlic Message will contain only one clove. However, the router will periodically bundle two additional cloves in the Garlic Message:



1. A Delivery Status Message, with Delivery Instructions specifying that it be sent back to the originating router as an acknowledgment. This is similar to the "reply block" or "reply onion" described in the references. It is used for determining the success or failure of end to end message delivery. The originating router may, upon failure to receive the Delivery Status Message within the

Hidden Services Manager

Hidden Services Manager

These are the local services provided by your router. By default, most of your client services (email, HTTP proxy, IRC) will share the same set of tunnels and be listed as "Shared Clients".

Status Messages

```
Starting tunnel A01 Omni*Web HTTPS...
Error listening for connections on /127.0.0.1 port 4444: java.net.BindException: Address already in use
Stopping client HTTP Proxy on 127.0.0.1:4444
Client error for 127.0.0.1:4444, check logs
Client ready, listening on 127.0.0.1:6668, delaying tunnel open until required
Client ready, listening on 127.0.0.1:7660
```

Refresh Clear

Global Tunnel Control

Tunnel Wizard Stop All Start All Restart All

I2P Hidden Services

Router Info

Version:	2.6.1-0
Uptime:	4 hours

Bandwidth in/out

3 Sec:	2.18 / 2.64 KBps
5 Min:	1.37 / 1.62 KBps
Total:	1.24 / 1.57 KBps
Used:	18.91 MB / 24 MB

Network: SymmetricNAT

I2P Services

Email
Torrents

I2P Hidden Services

Name	Type	Points at	Preview	Status	Control
A01 Omni*Web HTTPS	Standard server	127.0.0.1:443	No Preview		Stop
Destination: yo6sgmfq7pfvvp2e4kcuhjtfg7wfltt63igwcukhbmuqm6lu3a3a.b32.i2p **Description:** A01 Omni*Web HTTPS					
HTTP Phos tunnel	HTTP server	127.0.0.1:80	Preview		Stop
Destination: weor7pxsuxpkkbvh4sgta7hvoom5jrzemvy253y5norcamadujaq.b32.i2p **Description:** HTTP Phos tunnel					
I2P HTTPS Tunnel	HTTP server	127.0.0.1:7668	Preview		Stop
Destination: 6fjdbkxk7z3ykh5fauag3gopui7sx4sdzererl7jvsrdivsboqra.b32.i2p **Description:** I2P HTTPS Tunnel					
I2P webserver	HTTP server	127.0.0.1:7658	No Preview		Start
Hostname: mysite.i2p **Description:** My eepsite					

https://omnixtar.github.io/oxw/

Omni*Web::OXW

A Truly Decentralised Web ...

Omni*Web::OXW is an initiative to build a Truly Decentralised Web, *owned and operated by individual users and free software programmers.*

I2P

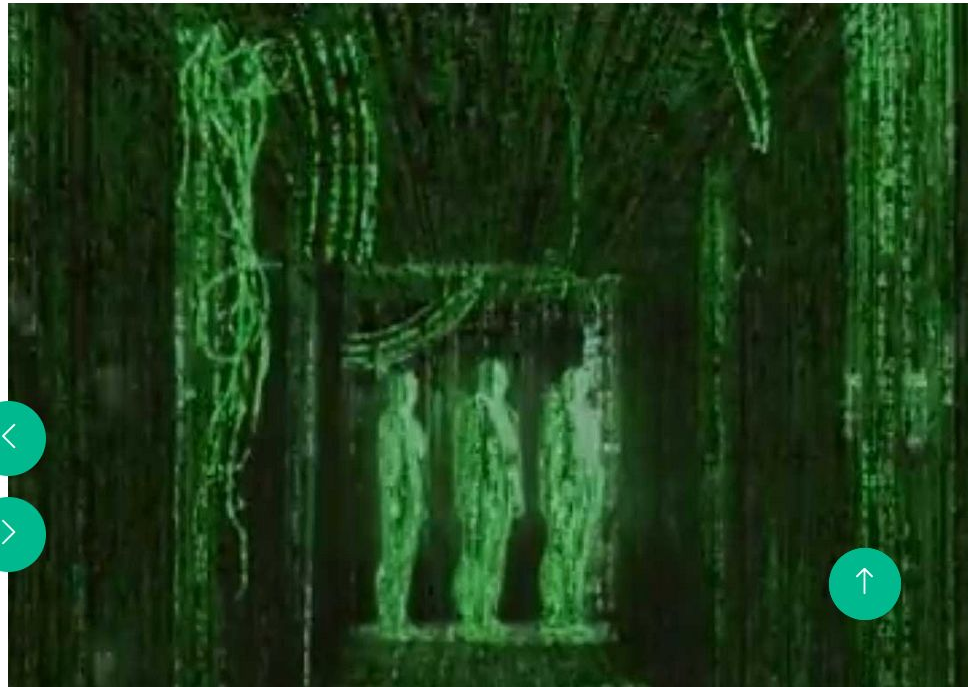
Firefox NGROK

Chrome NGROK

OXW is built using reliable and transparent open source software projects that began in 2003 or earlier.

Omni*Shell

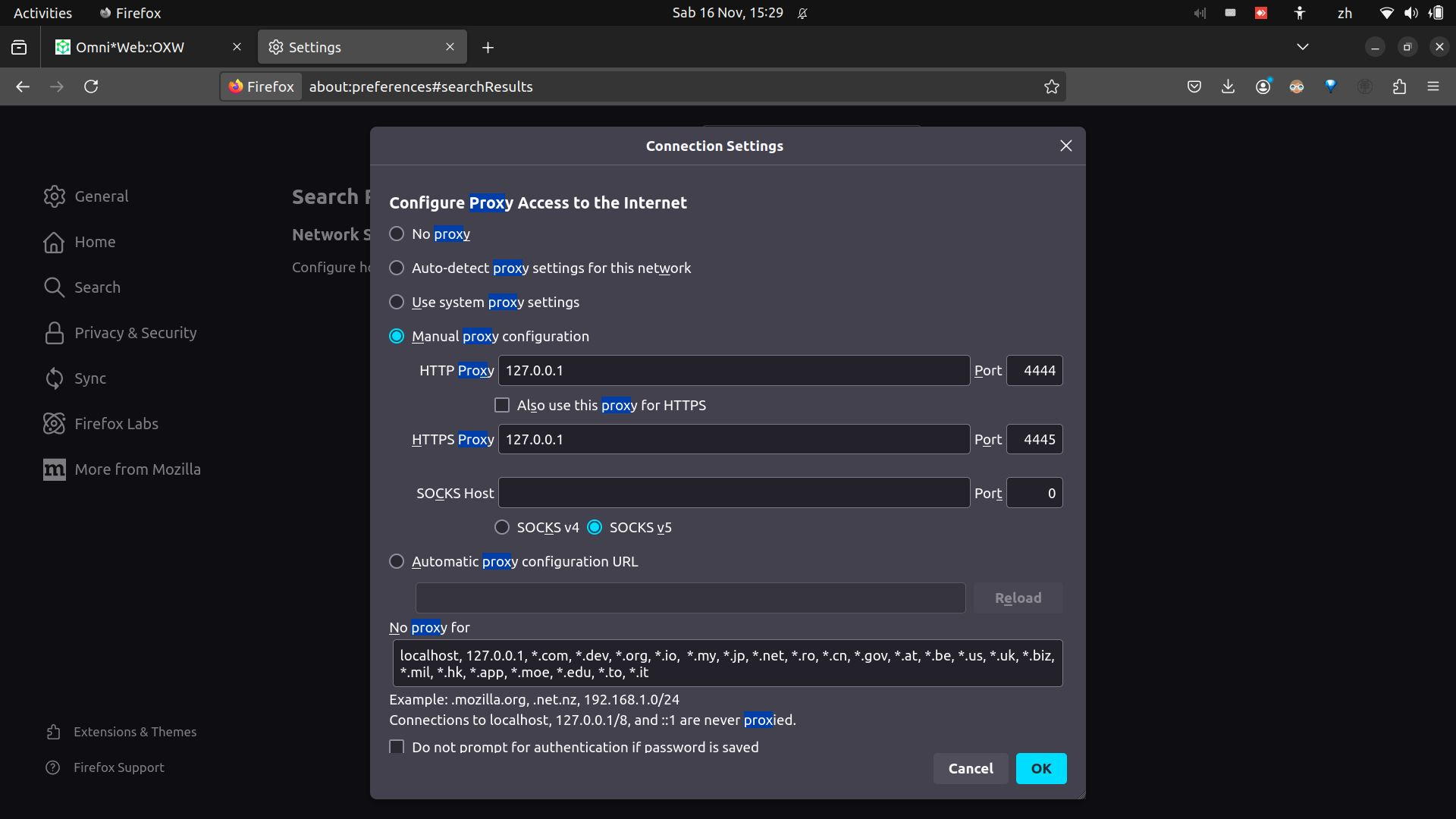
You may login to OXW test server via I2P (Invisible Internet Project) or NGROK tunnel connections. These are technologies designed to bypass Domain Name System, which devilishly prevented millions of programmers



```
hongwu@hongwu-Latitude-5480:~/i2p-2.6.1$ ./i2prouter
Usage:  ./i2prouter [ console | start | stop | graceful | restart | condrestart | status | install |
 remove | dump ]

Commands:
  console      Launch in the current console.
  start        Start in the background as a daemon process.
  stop         Stop if running as a daemon or in another console.
  graceful     Stop gracefully, may take up to 11 minutes.
  restart      Stop if running and then start.
  condrestart  Restart only if already running.
  status       Query the current status.
  install      Install to start automatically when system boots.
  remove       Uninstall.
  dump         Request a Java thread dump if running.

hongwu@hongwu-Latitude-5480:~/i2p-2.6.1$
```


Activities
Firefox
Sab 16 Nov, 15:29
zh
Omni*Web::OXW
Settings
Firefox
about:preferences#searchResults
General
Home
Search
Privacy & Security
Sync
Firefox Labs
More from Mozilla
Extensions & Themes
Firefox Support
Connection Settings
Configure Proxy Access to the Internet
No proxy
Auto-detect proxy settings for this network
Use system proxy settings
Manual proxy configuration
HTTP Proxy
127.0.0.1
Port
4444
Also use this proxy for HTTPS
HTTPS Proxy
127.0.0.1
Port
4445
SOCKS Host
Port
0
SOCKS v4
SOCKS v5
Automatic proxy configuration URL
Reload
No proxy for
localhost, 127.0.0.1, *.com, *.dev, *.org, *.io, *.my, *.jp, *.net, *.ro, *.cn, *.gov, *.at, *.be, *.us, *.uk, *.biz, *.mil, *.hk, *.app, *.moe, *.edu, *.to, *.it
Example: .mozilla.org, .net.nz, 192.168.1.0/24
Connections to localhost, 127.0.0.1/8, and ::1 are never proxied.
Do not prompt for authentication if password is saved
Cancel
OK

achow101 Merge #29833: i2p: fix and improve logs b27afb7 · 5 months ago

494 lines (414 loc) · 15.4 KB

Code Blame Raw

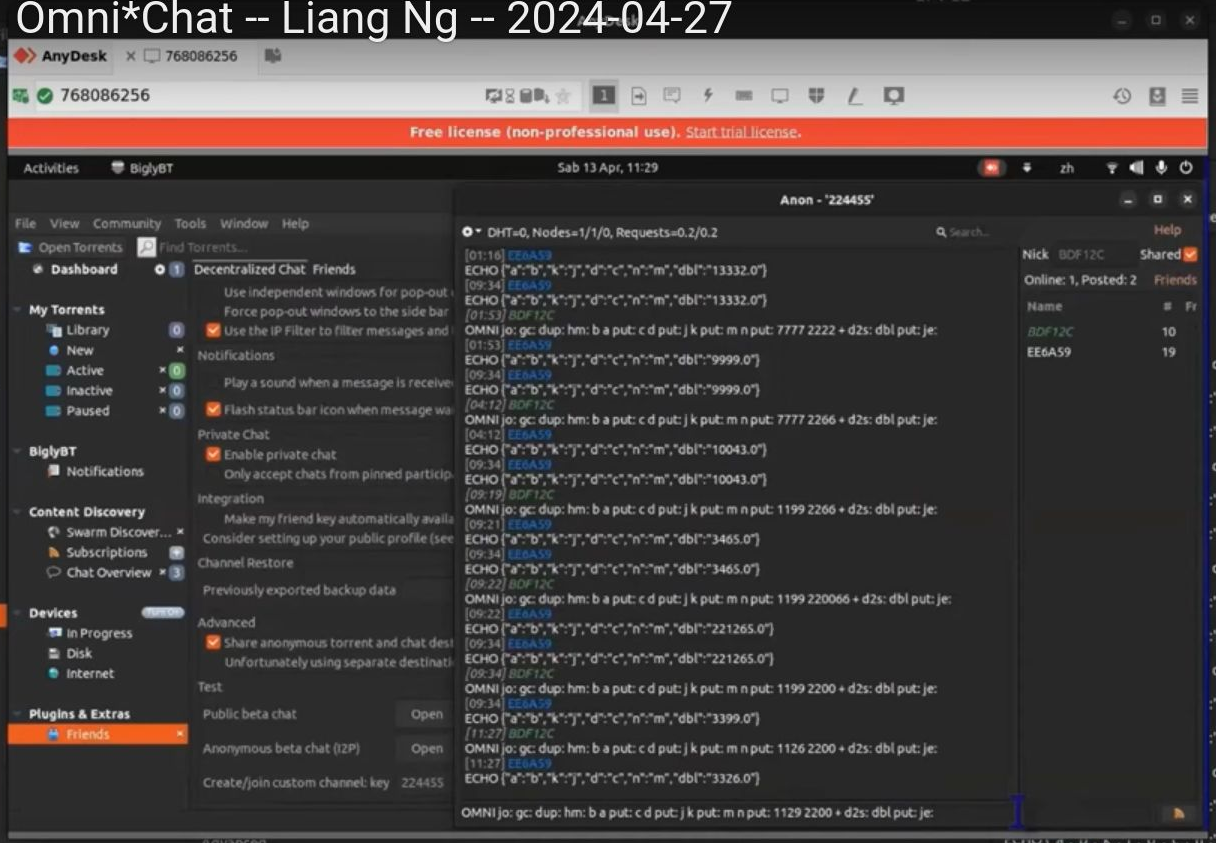
```
1   // Copyright (c) 2020-2022 The Bitcoin Core developers
2   // Distributed under the MIT software license, see the accompanying
3   // file COPYING or http://www.opensource.org/licenses/mit-license.php.
4
5   #include <chainparams.h>
```

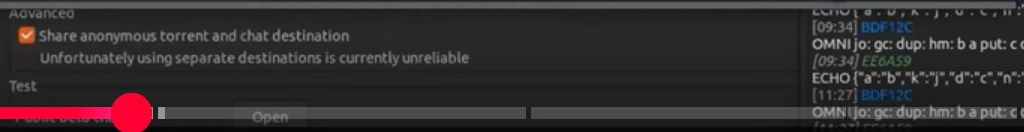
master bitcoin / src / i2p.cpp Top

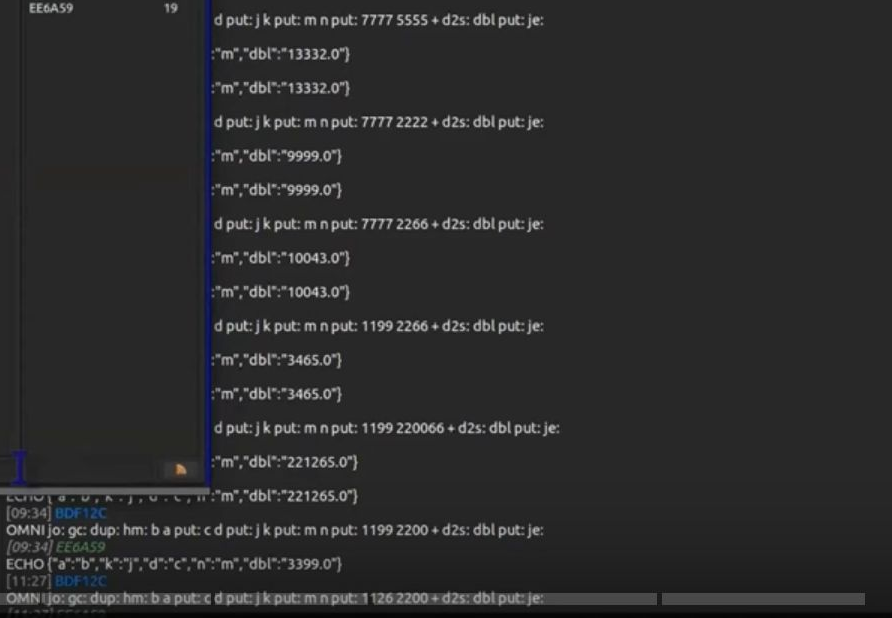
Code Blame Raw

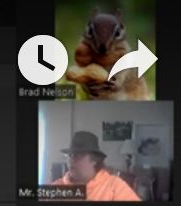
```
87      * @throw std::runtime_error if conversion fails
88      */
89     static CNetAddr DestBinToAddr(const Binary& dest)
90     {
91         CSHA256 hasher;
92         hasher.Write(dest.data(), dest.size());
93         unsigned char hash[CSHA256::OUTPUT_SIZE];
94         hasher.Finalize(hash);
95
96         CNetAddr addr;
97         const std::string addr_str = EncodeBase32(hash, false) + ".b32.i2p";
98         if (!addr.SetSpecial(addr_str)) {
99             throw std::runtime_error(strprintf("Cannot parse I2P address: \"%s\"", addr_str));
100        }
101
102        return addr;
```

Omni*Chat -- Liang Ng -- 2024-04-27









SSR-2024-04-13-11.29.04.mkv

4:03 / 24:32 • Introduction ›

```
21  public class Hash extends SimpleDataStructure {
106     }
107
108     /**
109      *  For convenience.
110      *  @return "{52 chars}.b32.i2p" or null if data not set.
111      *  @since 0.9.25
112      */
113     public String toBase32() {
114         if (_data == null)
115             return null;
116         return Base32.encode(_data) + ".b32.i2p";
117     }
118
119     /**
120      *  @since 0.9.17
```