btcd: An alternative full-node implementation written in Go

Dave Collins, Senior Engineer

About Me

- My background:
  - Over 2 decades programming experience
  - Open source enthusiast and contributor
  - Low-level proprietary assembler on military hardware
  - Zero-knowledge remote cryptographic storage
  - Distributed systems
  - Databases
- Currently:
  - Senior Engineer at Conformal Systems
  - Lead Developer on btcd

Why Go?

- Integrated test infrastructure
- Compiles to native code
- No active memory management
- Standard formatting via go fmt
- Platform independent code
  - Example: irc member confirmed btcwire worked on Plan9 and was able to talk to a live bitcoind node in less than a day
- Great concurrency support (CSP)
- Crash resilient
- Built-in profiling and documentation facilities

How Btcd Makes Use of Go

- Packages
- Channels
  - Transaction script verification
  - CPU mining
- Interfaces
  - Database
  - Addresses
- Testing
  - Extremely high test coverage using built-in Go testing facilities
- Dependency Fetching
- Build Infrastructure

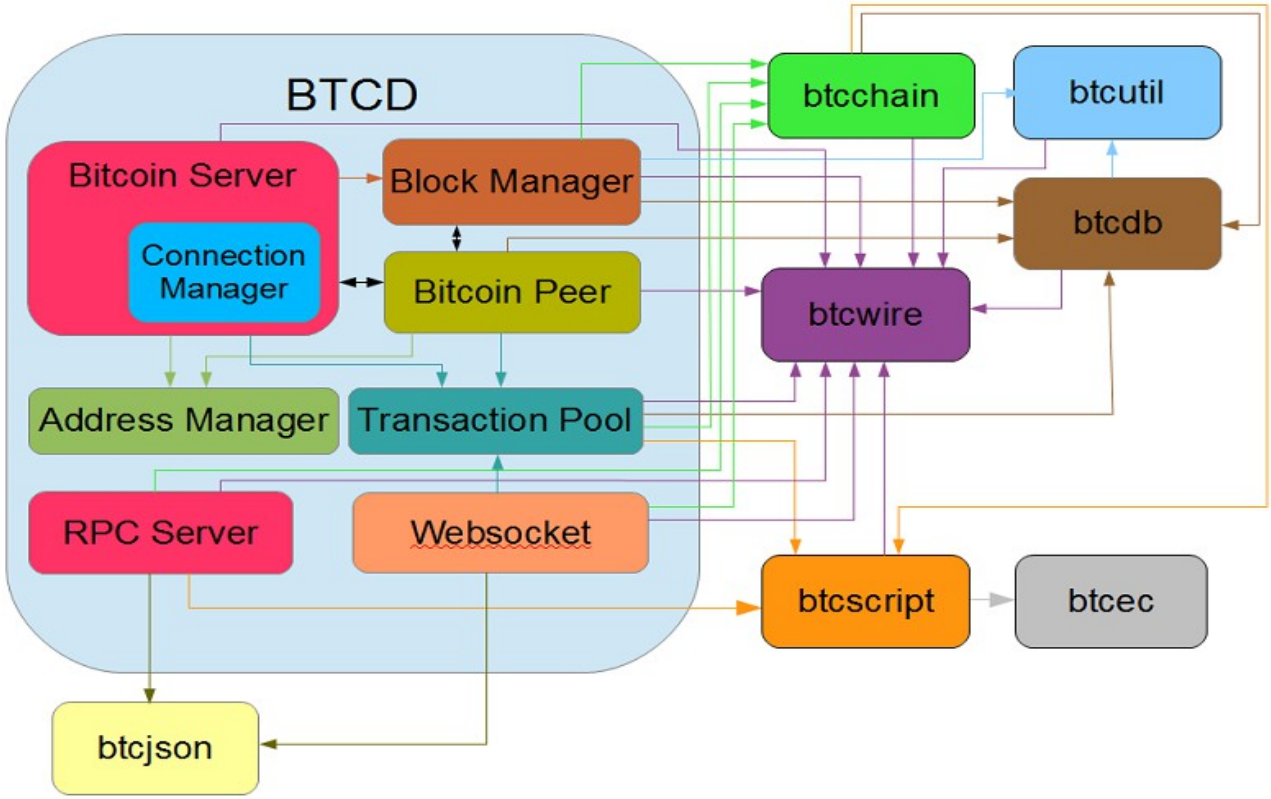
Go Weaknesses

- Lack of const for returning or accepting
- Currently unable to create shared libraries callable from other languages
- Variable shadowing with no warnings
- Defer is only function scoped
- Garbage collector can be slow
  - Can be mitigated by paying attention to memory management
  - Improving with each version and more slated for Go 1.5

Go Ecosystem Improvements

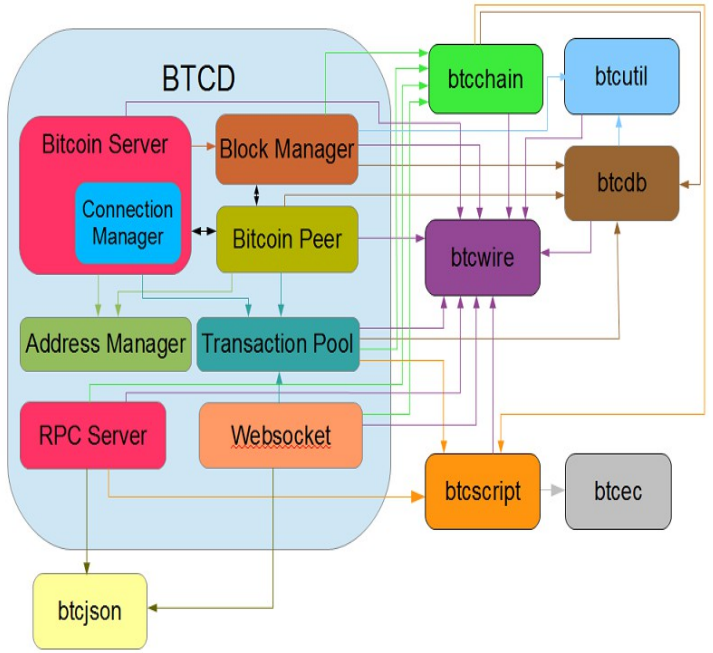
- Worked with core Go dev to get:
  - ECDSA support in TLS
  - Optimized sha256 assembly implementation for i386 and amd64
- Discovered and report various memory leaks
  - Append, Not setting final slice element to nil
- Go-spew
  - Deep pretty printer for Go data structures to aid in debugging
- Go-xdr
- Bitcoin packages for working with the entire Bitcoin stack!

Btcd Architecture Overview



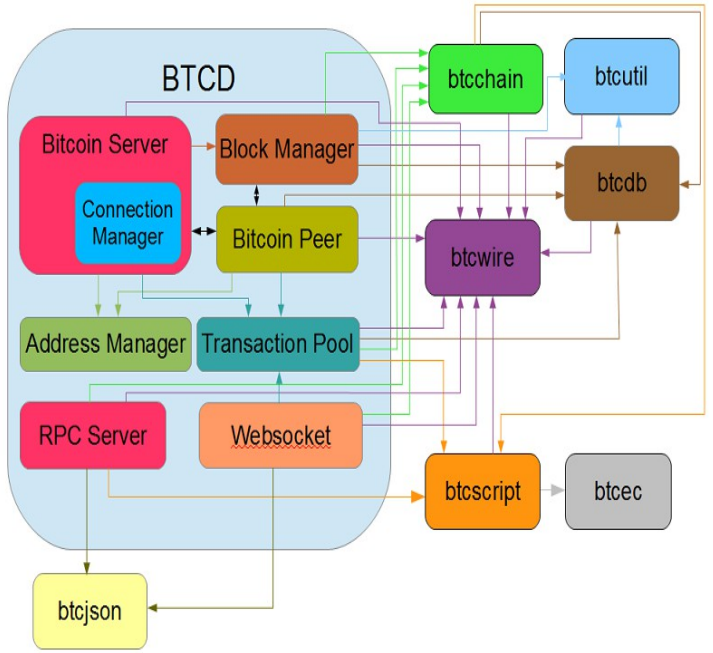
Individual Package Breakdown

- btcwire
  - Core package on which everything builds
  - All wire protocol serialization/deserialization
- btcnet
  - Provides network parameters
  - Support for custom network registration
- btcdb
  - Provides database interface for blocks and txns
  - Supports multiple backends (leveldb, memdb)



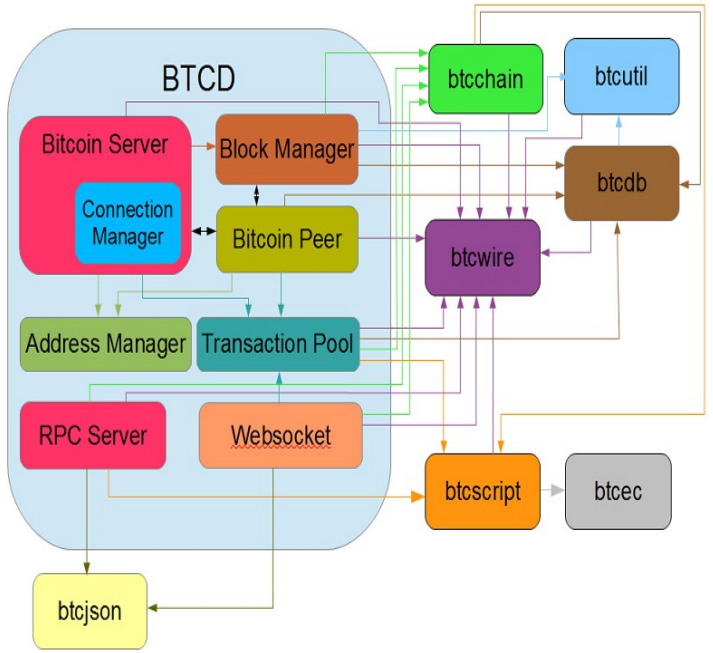
Individual Package Breakdown (Cont)

- btcec
  - Provides highly optimized secp256k1
  - Sign, verify, and serialize pubkeys/sigs
  - Compact signing and recovery
- btcscript
  - Executes and validates tx scripts
  - Creation of multi-signature p2sh scripts
  - Supports building custom scripts
- btcchain
  - Chain consensus rules
  - Difficulty target conversion
  - Support for checkpoints defined by btcnet



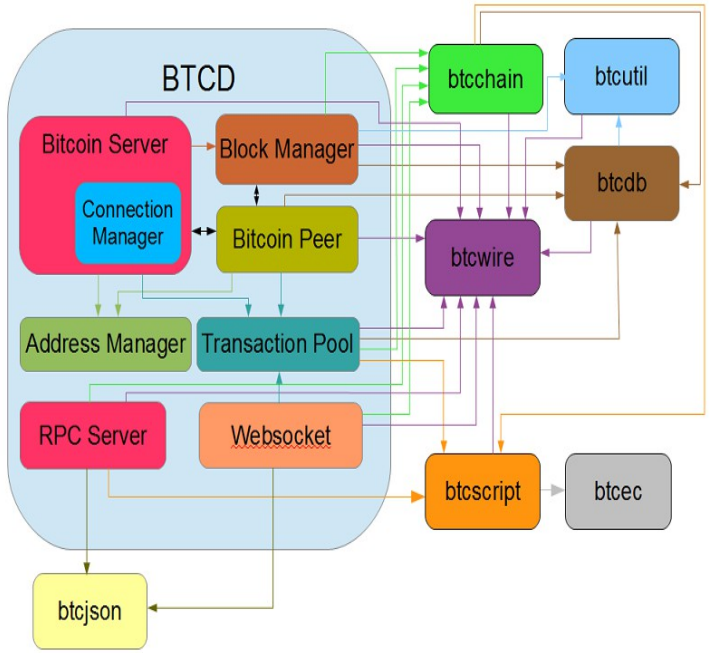
Individual Package Breakdown (Cont 2)

- btcutil
  - Modified Base 58 encoding/decoding
  - Address encoding/decoding for all networks
    - Pubkeys
    - Pubkey hashes
    - Script hashes (Important for multi-sig)
  - Wallet Import Format (WIF)
  - Block and Tx convenience wrapper types
- btcutil/hdkeychain
  - Support for BIP032 HD extended keys
  - Cryptographically secure seed generation
  - Seamless integration with btcec and btcutil



Individual Package Breakdown (Cont 3)

- btcrpcclient
  - Higher level API around btcjson/btcws
  - Seamless integration with other packages such as btcwire, btcutil, and btcec
  - Support for btcd extensions and websockets
  - Robust with automatic reconnect, notification re-registration, and command reissue



Future Plans

- Wallet support for multi-account deterministic hierarchical keys
- Database optimizations
- Concurrent block downloads during initial download
  - Already support headers first, but currently only uses a single sync peer
- Support for more advanced features such as stealth addresses and prefix filters

Questions?