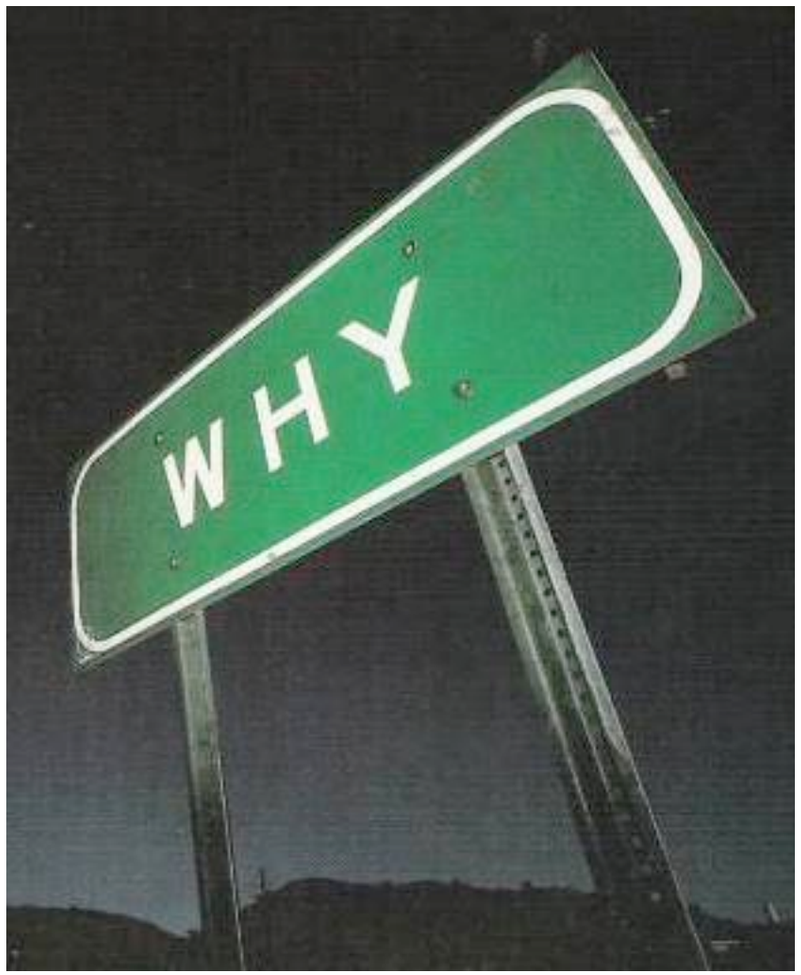
πp

Anant Narayanan

August 23, 2010

What is πp?

- A fast, simple, distributed, reliable, versioned, caching network file protocol


WHY

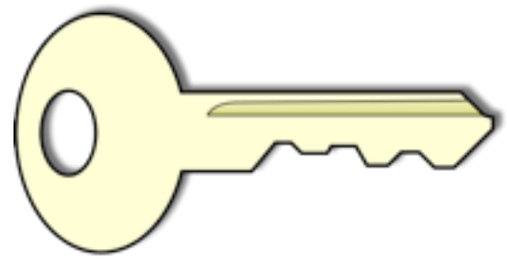
Another Protocol?

- The current design of the internet is based on communicating peers
- Every time content is accessed, clients are individually handed data from the server
- Can this approach really scale?

Data Has Changed

- HTTP over TCP does well for the types of data it was designed to transfer
- HTML5 supports video, but is HTTP over TCP the best way to transport it?

Authentication

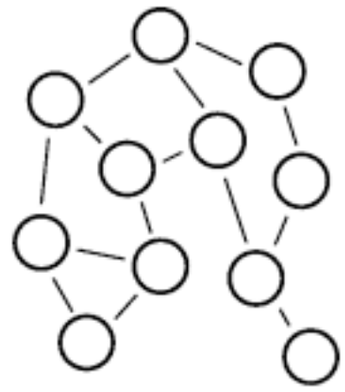


- Access control in any modern web application is *ad hoc* and relies on methods like browser cookies
- HTTP does support basic forms of authentication (of both client & server) but nobody seems to be using it!

Anonymity

- Almost every corporate network uses firewalls to filter all traffic not on port 80, and even HTTP is subject to deeper packet inspection
- This can't go on forever, unless we change the *way* in which content is distributed

Decentralization



- Autonomy is a defining feature of the Internet
- Yet, we observe large amounts of aggregation of user data towards a few third party services (Google, Facebook)

Sharing

- The best way to share something today is to store data on someone else's server
- This needs to change

Synchronization

- We're moving away from the paradigm of several people sharing a single computer towards several devices serving a single person
- It's just a better user experience to "carry your data with you"

Existing Technology

FTP

- Very limited in use, no versioning or file metadata support
- Prone to bounce attacks
- Little scope for caching

Coda

- Complex (~90k lines of C++ code)
- Dynamic files unsupported
- No support for versioning despite strong file sharing semantics

NFS

- Also complex in implementation though there are several interoperable choices
- No support for dynamic or device files
- Concurrent access for shared files is disallowed

SMB/CIFS

- Proprietary
- No versioning support
- Single reference implementation
- Only works over reliable transport (NetBIOS and TCP)

9P2000/Styx

- No support for pipelining requests
- No support for rich file metadata
- Only works over reliable transport

How?

Everything is a file!

- We take the approach of representing the entire internet as a large distributed filesystem

Goals

Simplicity

- Both in specification and in implementation
- Limit feature set to cover 90% of current use-cases

Flexibility

- This can mean many things, but a few of them are:
  - Don't limit ourselves to a username/password authentication paradigm
  - Extensible file open modes
  - Client endpoint portability

Reliability

- Be only as reliable as is needed
- This means not relying on TCP for *everything*
- Data types likes video work much better when the client has more control over what pieces (frames) it needs and when

Metadata

- Almost every operating system implements arbitrary metadata
- Enables a large set of applications:
  - Better search and indexing
  - Eliminates the need for *ctl* files
  - Wacky: Facebook-esque comments!

Versioning

- Simple form of backup
- Automatically provides an audit trail
- Greatly simplifies caching content
  - The problem is reduced to knowing what the latest version of a file is

Distributed-ness

- Simple form of backup
- Automatically provides an audit trail
- Greatly simplifies caching content
  - The problem is reduced to knowing what the latest version of a file is

Design

Messages

- Request/Response model
- 10 Basic Operations:
  - Tsession, Tattach, Tclunk
  - Topen, Tclose, Tread, Twrite
  - Tcreate, Tremove, Tflush

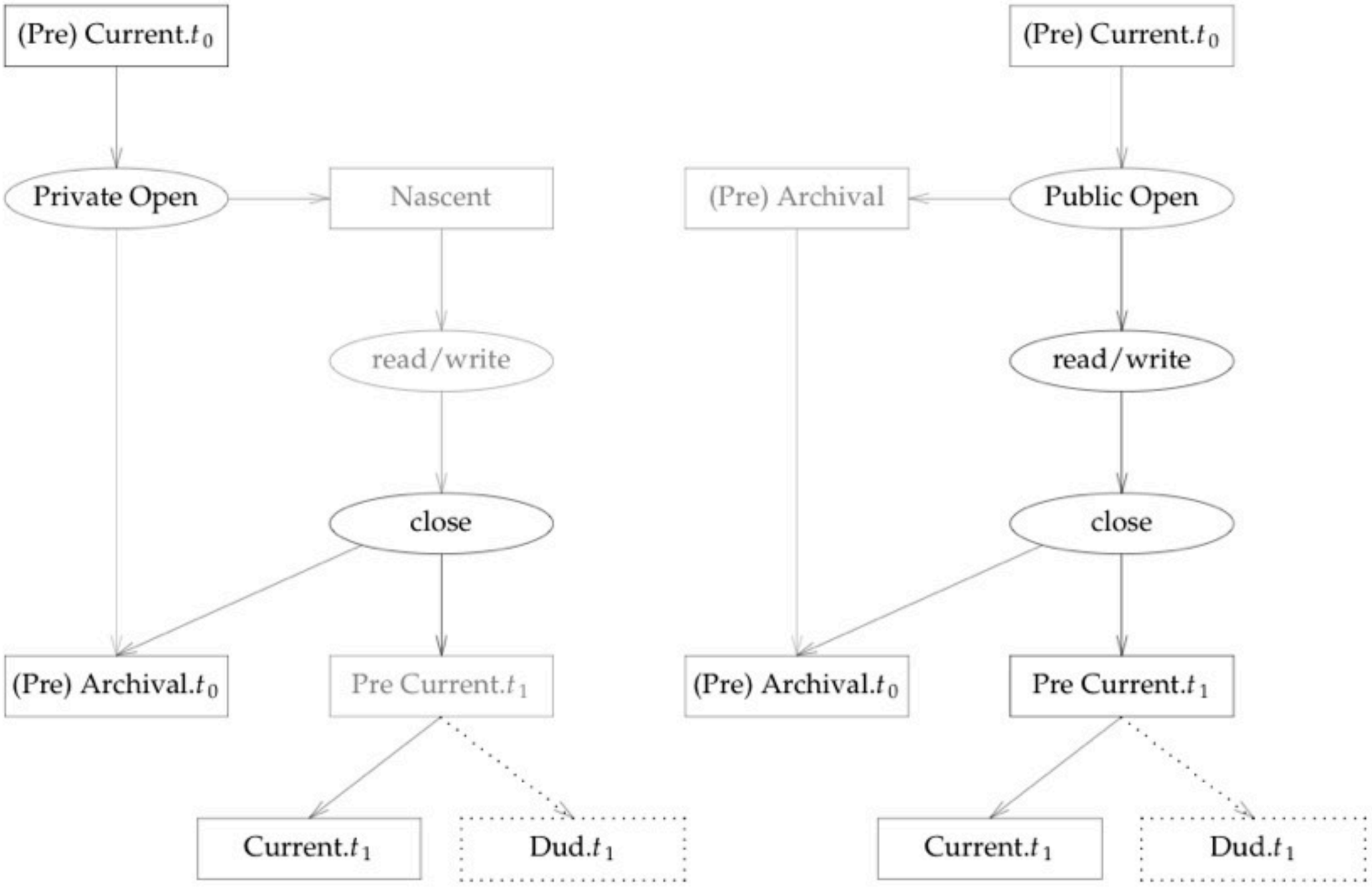
Messages

- Responses are prefixed with R instead, with the exception of Rerror
- A single message may contain multiple requests or responses
- Responses are always in the order of the requests

Versions

- All non-dynamic files are versioned
- Versions are immutable and committed on file close
- A ‘version’ is simply a 64-bit timestamp

Two Commit Types



Message Layout

5 data types: u16int, u32int, u64int, data, string

{hdr:data}{len:u32int}{id:u32int}{tag:u32int}K{O1...On}

Session ID Exchange

```
Tsession
{csid:u32int}{afid:u32int}{msize:u32int}{options:string}
```

```
Rsession
{ssid:u32int}{afid:u32int}{msize:u32int}{options:string}
```

Authentication

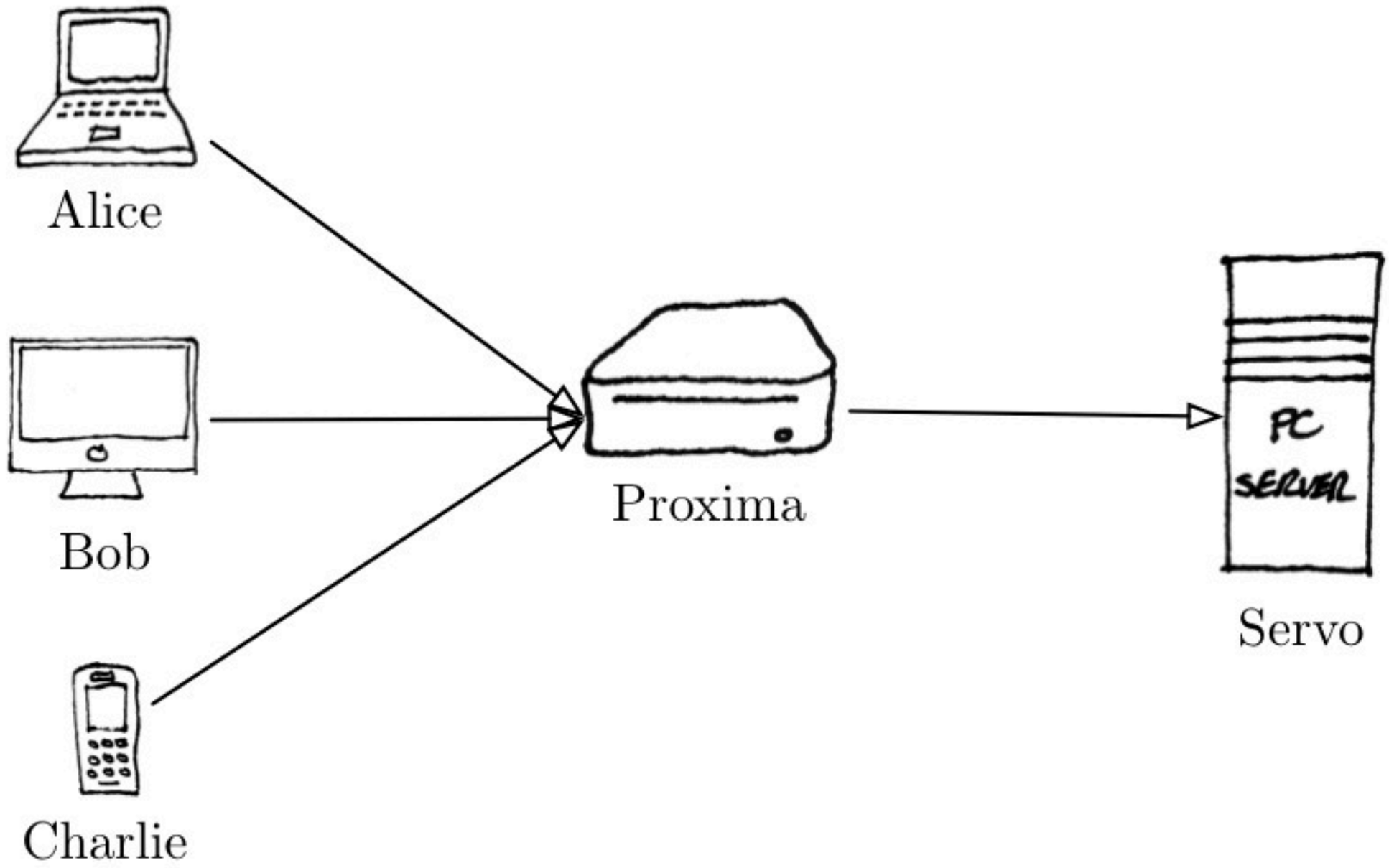
- Exact scheme used is left to the client/ server to decide
- The protocol provides an 'afid' that the server will accept regular file operations (read and write) on to execute a particular authentication mechanism
- Encryption may also be prepared this way (key exchange)

Proxying

`Tattach`
`{fid:u32int}{afid:u32int}{uname:string}{aname:string}`

`Rattach`
`{afid:u32int}`

Proxying



Session Close & Flush

```
Tclunk
{ssid:u32int}

Rclunk
{}

Tflush
{tag:u32int}

Rflush
{}
```

File Open

Topen

{fid:u32int}{nfid:u32int}{path:string}{mode:string}

Clone

nfid = fid

Walk

fid = fid/path

Open

File set to open with ‘mode’ and cannot be walked

Ropen

{ftype:u32int}{version:u64int}{len:u64int}

File Close

Tclose
{fid:u32int}{commit:u16int}

Rclose
{version:u64int}

Read & Write

```
Tread
{fid:u32int}{offset:u64int}{count:u32int}{attrs:string}
Rread
{dat:data}

Twrite
{fid:u32int}{offset:u64int}{dat:data}{attrs:string}
Rwrite
{count:u32int}
```

Metadata

- Manipulated using Twrite and read using Tread by use of 'attrs'
  - '*' implies all attributes
  - '#' implies a predefined set of values
- Key-value pairs are one per line, appropriately quoted

Create & Remove

```
Tcreate
{fid:u32int}{name:string}{perm:u32int}{mode:string}
{ftype:u32int}

Rcreate
{version:u64int}

Tremove
{fid:u32int}

Rremove
{}
```

Did It Work?

Generator

- Operations and arguments were changing fast during the design
- 800-line code generator takes a 125 line JSON description of the protocol and creates Go and C versions of a message parsing library
- 300-line Go server helper builds on this to provide UDP and TCP transports

Quick Test

File Download (Average over 10 attempts)

1 x 600MB	
Protocol	Time
πp	46.970s
FTP	47.195s
HTTP	51.464s
NFS	44.945s

600 x 1MB	
Protocol	Time
πp	32.432s
FTP	1m18.619s
HTTP	1m26.156s
NFS	44.945s

Some Ideas

- RPC (metadata instead of *ctl*)
- Wikifs (flexible open modes)
- Video Stream (UDP transport/Tflush)

Thank You!

Leasing

```
Tlease
{fid:u32int}
Rlease
{expires:u64int}

Trenew
{}
Rrenew
{}

Trevoke
{fid:u32int}
Rrevoke
{}
```

Reliability

`Tack`
`{tag:u32int}`

`Tenq`
`{tag:u32int}`

`Renq`
`{tag:u32int}`