ECP - A Protocol for P2P CTree Clouds

By Philip Horger

May 31, 2011

Part I
Introduction

ECP in its modern sense was "invented" fairly recently, but it's been in the works for a long time, with a lot of research dead ends. This has led to a lot of misinformation, stale information, etc. That's what this introduction intends to help clear up. It does not cover what BCP is, or CTree, or what either is used for. Those resources exist elsewhere publicly, so we're assuming that if you're reading this, you're already familiar with them, or you're familiar with Google.

The most notable dead end of the whole thing was that ECP was going to be an extension of the BCP message set. In fact, that's where the name comes from: Extended CTree Protocol (as opposed to Basic CTree Protocol). It was intended to add topology management, routing, authentication, and other such niceties to BCP, by adding a few more message types. We were so sure about this part of it... well, we told everybody that's how it was gonna work. *Everybody.*

Modern ECP is very fundamentally different from this. It's a document standard that sits on top of a generalized DHT model. Plug in whatever DHT tech you want underneath it - Kademlia, Chord, Pastry... any tech where keys can be arbitrarily chosen (not a hash of the value, for example) can be used. Live connections are handled with direct P2P BCP over TCP connections.

As well, there are some optional parts on top that you can use to encourage "good behavior" in the DHT network. You are highly encouraged to use memristor limiting, which is a network-wide option. Viral survival should in theory be a very powerful data replication algorithm in concert with memristor limiting, which is a client-level option.

Part II
Terminology

1 Object

1.1 In general

An object has 4 properties: key, name, typecode, and value. It represents a single entity in the DHT. The key is always the typecode plus the SHA512 hash of the name. The only variable that can change is the value.

All objects are meant to be preprocessed by PGP. They might take advantage of this, they might not.

The primary thing that ECP does is standardize the contents of these objects according to object type. This in some cases includes restrictions or requirements for certain PGP features, like signing or encryption.

1.2 Structure

A document is, generally, preceded by a "permission" header based on the document name, with a delicious PGP center.

2 DHT

Distributed hash tables are a general category of swarm-based storage that, with a little abstraction, allows you to treat the sum total of all client hosting as a single, queryable database. Popular ones often come with APIs and libraries for a variety of languages.

ECP doesn't dictate what DHT algorithm or library you use, just what you store on it. But you do need to consider the following:

1. Arbitrary keys. Any DHT that doesn't let you set arbitrary keys for your values (such as the Content-Addressable Network) simply aren't compatible with ECP at all.

2. The ability to throttle, drop, and monitor bandwidth for a connection. This is not a strictly necessary feature for basic ECP, but are the minimum requirements for the optional feature "memristor routing," which provides resillience against "spammy" and "leechy" nodes - superbly useful if you're expecting DoS or DDoS attacks on your network.

3. Peer queryability. For DHTs that do not make much in the way of guaranteed robustness, a less-agressive, memristor-aware variation of viral survival can make documents less susceptible to deletion through unpopularity. This lets you use other metrics (such as paying for hosting with hosting). It does require that your network allows you to query your peers for whether they contain a given object, or determine that algorithmically, so that all objects can be backed up/hosted on at least n machines.

3 MEL

An MEL, or multiple-encryption list, is a way to restrict a small piece of information to a list of participants based on their public keys.

The source text (information to be encrypted) along with a magic string is encrypted with each public key individually. Given the whole list, any user with the correct private key can find the decrypted source text by using their key to decrypt each line indidually and searching for the magic string as a substring.

Since it's not hard to do the equivalent of find-and-replace on the correct answer, there's no reason the magic string can't be at any arbitrary position. This means we can increase security by using a different random substring position for every line when we encode, which increases the security of the other participants if anyone's private key is cracked.

Part III
Object Type Standards

Auth: an object that stores the symmetric keys of documents in a key:value (or really, name:value) type of format. It is encrypted with its own symmetric key, signatures optional. Typecode 0001.

AuthRoute: an MEL of the symmetric key of an Auth, encoded in the keys of all the people with access to those documents. Must not be encrypted or signed with PGP. Has the same name as the Auth it allows access to (so the only difference between keys is the typecode). Typecode 0000.

DocMeta: a JSON object that stores metadata information about a document. This includes participant information, live node locations, and other information. The structure is still under development at the time of this writing. Shares the same name as the document it describes, is encrypted with the document's symmetric key. Typecode 0010.

DocSegment: a JSON-serialized CTree layer. Uses the document's symmetric key for encryption. Typecode 0011.

NameValidation: the public key associated with a name. This allows storage nodes to validate and version-compare DHT insertions. Its self-validation mechanism is yet to be determined.