HGFRR: Hidden Geographical Fractal Random Ring

Anonymous Authors

Abstract

This is the abstract.

1 Introduction

This is the introduction.

Flow:
start from talking about current blockchain systems, transaction rate is getting increasing
→ problem is: current blockchain systems are based on kademlia (dht) structure, which is efficient on node discovery and data look up, however, broadcast suffers traffic congestion and inefficient convergence (gossip)
→ many improvements are made on gossip, not quite efficient, still suffers bandwidth problem
→ less p2p network are designed for blockchain, talk about p2p networks in other domains
→ analysis blockchain needs and those p2p networks does not fit
→ moreover, current blockchain systems didn't address geographical feature
→ moreover, current blockchain systems didn't address security problem from the p2p layer
→ talk about hgfrr, give a overview, analysis result, implementation overview, evaluation result
→ summarize our feature & contributions
→ paper structure overview

2 Background and Motivation

This is the background and motivation.

2.1 Peer-to-Peer Overlay Networks

Talk about the overview on various peer-to-peer networks. Structured vs unstructured.

2.1.1 Node Discovery

Talk about the node discovery in p2p networks. DHT. Chord, CAN, Tapestry, Pastry.

2.1.2 Broadcast

Talk about the broadcast in p2p networks. Various version of Gossip. Broadcast in tree-based network.

2.2 P2P Networks in Blockchain Systems

Talk about current p2p networks in a blockchain system, blockchain system's demand.

2.3 Trusted Execution Environment (TEE)

Talk about trusted execution environment, Intel SGX.

2.4 Design Motivation

Talk about the design motivation, kind of combining the two.

3 HGFRR Design

This is the design.

3.1 Design Principles

Talk about the basis of the design.

3.2 Topology

Talk about the p2p network structure.

3.2.1 Structure Formation

Talk about the p2p network formation. boostrap.

3.2.2 Structure Maintenance

Talk about the p2p network maintenance.

3.3 Broadcast

Talk about broadcast mechanism.

3.4 Security Consideration

Talk about the usage of Intel SGX, fake messages, contact node election.

4 Proof and Analysis

This is the proof and analysis.

4.1 Proof of Broadcast Performance

give a proof of time complexity and message comlexity of broadcast, node join and contact node election. Compare to current works.

4.2 Security and Robustness Analysis

analyze of fault tolerance, anonymity.

5 Implementation Details

This is the implementation details.

How we implement the system. and the architecture of the codebase.

6 Evaluation

This is the evaluation.

how we evaluated hgfrr.

6.1 Evaluation Setup

talk about how we set up the evaluation

6.2 Ease of Use

use kad+gossip to substitute eth

6.3 Performance Improvements

talk about the performance improvement in terms of convergence time, message complexity.

6.3.1 Broadcast

broadcast performance

6.3.2 Robustness and Scalability

fault-tolerance and scalability of broadcast

6.3.3 Contact Node Election

performance of contact node election

6.3.4 Node Join and Leave

performance of node join and leave, which may lead to structure change

6.4 Security Evaluation

wireshark analysis to show the anonymity of contact nodes

7 Related Work

This is the related work.

P2P Network in Ethereum Talk about kademlia+gossip in ethereum.
Broadcast in DHT Talk about a work working on broadcast in dht.
more to be added and compared.

8 Conclusion

This is the conclusion.

Give a conclusion on background, motivation, overview of hgfrr, feature and analysis result. future work.

9 Acknowledgments

A polite author always includes acknowledgments. Thank everyone, especially those who funded the work.

10 ATC Template For Reference

```
int wrap_fact(ClientData clientData,
              Tcl_Interp *interp,
              int argc, char *argv[]) {
    int result;
    int arg0;
    if (argc != 2) {
        interp->result = "wrong # args";
        return TCL_ERROR;
    }
    arg0 = atoi(argv[1]);
```

Figure 1: Wonderful Flowchart

```
    result = fact(arg0);
    sprintf(interp->result,"%d",result);
    return TCL_OK;
}
```

Now we're going to cite somebody. Watch for the cite tag. Here it comes [**?**]. The tilde character (˜) in the source means a non-breaking space. This way, your reference will always be attached to the word that preceded it, instead of going to the next line.

More fascinating text. Features[1] galore, plethora of promises.

11 This Section has SubSections

11.1 First SubSection

Here's a typical figure reference. The figure is centered at the top of the column. It's scaled. It's explicitly placed. You'll have to tweak the numbers to get what you want.

This text came after the figure, so we'll casually refer to Figure 1 as we go on our merry way.

11.2 New Subsection

It can get tricky typesetting Tcl and C code in LaTeX because they share a lot of mystical feelings about certain magic characters. You will have to do a lot of escaping to typeset curly braces and percent signs, for example, like this: "The `%module` directive sets the name of the initialization function. This is optional, but is recommended if building a Tcl 7.5 module. Everything inside the `%{, %}` block is copied directly into the output. allowing the inclusion of header files and additional C code."

Sometimes you want to really call attention to a piece of text. You can center it in the column like this:

`_1008e614_Vector_p`

and people will really notice it.

The noindent at the start of this paragraph makes it clear that it's a continuation of the preceding text, not a new para in its own right.

Now this is an ingenious way to get a forced space. `Real *` and `double *` are equivalent.

Now here is another way to call attention to a line of code, but instead of centering it, we noindent and bold it.

`size_t :  fread ptr size nobj stream`

And here we have made an indented para like a definition tag (dt) in HTML. You don't need a surrounding list macro pair.

> `fread` reads from `stream` into the array `ptr` at most `nobj` objects of size `size`. `fread` returns the number of objects read.

This concludes the definitions tag.

11.3 How to Build Your Paper

You have to run `latex` once to prepare your references for munging. Then run `bibtex` to build your bibliography metadata. Then run `latex` twice to ensure all references have been resolved. If your source file is called `usenixTemplate.tex` and your `bibtex` file is called `usenixTemplate.bib`, here's what you do:

```
latex usenixTemplate
bibtex usenixTemplate
latex usenixTemplate
latex usenixTemplate
```

11.4 Last SubSection

Well, it's getting boring isn't it. This is the last subsection before we wrap it up.

Notes

[1]Remember to use endnotes, not footnotes!