Stephan Karpischek Follow
Digital strategy and blockchain
yesterday · 6 min read

Flight Delay Dapp — lessons learned

Results and findings from putting a smart contract for sharing risks on the blockchain for two weeks.

Wow, what a blockchain week! Devcon two and blockchain summit in Shanghai were great. We are still processing all the input and impressions.

Christoph and Stephan at the ether.camp party at Bar Rouge (Photo by Griff Green)

On 8 September, more than three weeks ago, we launched a DApp on the Ethereum mainnet with a user interface at fdd.etherisc.com. The DApp enabled users to share the risks of delayed or canceled flights on the Ethereum blockchain. Read our announcement for details.

First, we want to thank all the brave users who applied for a policy: Our main goal is to learn from you; and with your help we can further improve the experience of using smart contracts. Thank you for using the Flight Delay DApp, and thank you for being so verbose about it!

The numbers

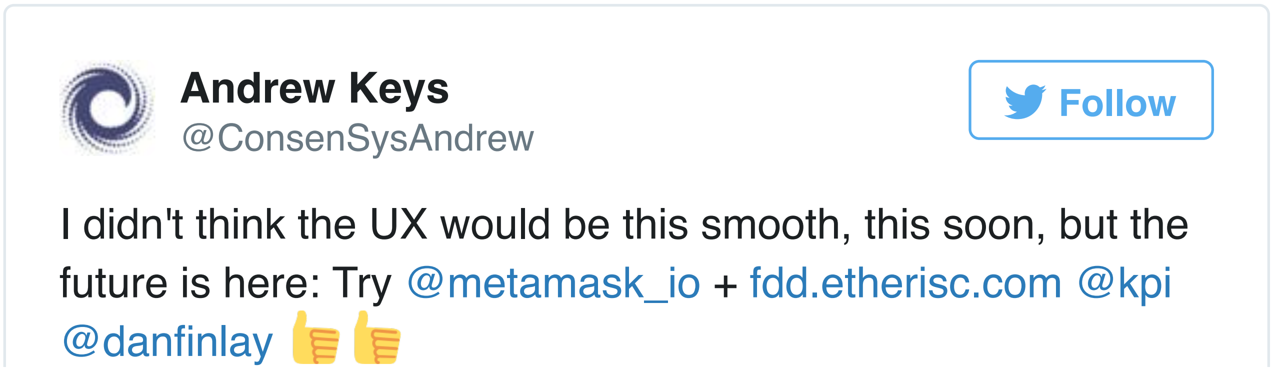
The DApp accepted policies for more than two weeks from 8 September to 25 September with a short interruption between 18 and 20 September in which we updated the smart contract to fix a critical bug (details below). In this time the contract received a total of 40 policy applications of which 31 (77.5 %) were accepted, 7 declined, and two could not be processed. The sum of all premiums paid in was 29.39 ETH.

On the mainnet a total sum of 30.8 ETH was paid out to 6 policies. This is pretty close to the zero-sum game we hoped to create, which is surprising, given the relatively small number of policies. In summary, we are 1.4 ETH short now (and a ton of experience richer).

On the morden testnet there was even more activity: 63 policies in total until 25 September. We deployed the contract on testnet again without a deadline, and people continue to use it, with 15 new policies so far.

Reactions

We really enjoyed the great discussions with users and interested parties at the conference. Thank you all for your interest and support!


Andrew Keys
@ConsenSysAndrew

I didn't think the UX would be this smooth, this soon, but the future is here: Try @metamask_io + fdd.etherisc.com @kpi @danfinlay 👍👍

Andrew Keys from ConsenSys liked the user experience. Thank you, Andrew!

We also read valuable feedback and criticism on reddit, Hacker News, and Twitter. Most interesting for us was the discussion about incentives to delay flights on purpose, see e.g. these comments on reddit and Hacker News.

Andrew Quentson interviewed us for an article on cryptocoinsnews. We quote his poetic description of the disintermediation that is taking place; what we believe to be the major driving force for many more interesting decentralized applications to come:

> *“Through an entry form, what previously required, and still, currently, requires, whole insurance departments and even skyscrapers, is turned into just 1s and 0s.”*

(Andrew Quentson for cryptocoinsnews)

The bugs

Two days after launch, one callback transaction from oraclize threw a surprising error. Our investigation showed that the reason was a race condition on the blockchain: the callback transaction was mined 14 blocks before the initiating request. Enabling the contract to handle this would have meant a major change, so we decided to wait and see if it happens again (and it did once more). The proper fix is to wait with the callback until the initial request has been mined with a few confirmations; we are looking forward for oraclize increasing the number of confirmations to wait or even better making this a configurable option.

Then, about one week after the launch, we monitored a strange transaction applying for a new policy for flight JL-5041, departure on 17 September. Neither the airport nor the departure day could have been selected in the web interface.

Investigating this further, we realized a critical bug: The contract would even accept applications for policies with a departure date in the past, so in fact it would allow betting on a flight delay after the flight had actually landed. This could have been easily abused to drain the contract. We decided to activate emergency mode: This locks down the contract, stops accepting new policies and freezes all payouts. The next version of the contract fixed the bug and was deployed a bit later. We manually paid out for four delayed flights.

Lessons learned

“Don’t call it insurance!”—Before the launch, we were warned by several parties that launching an insurance on the blockchain would very likely be considered illegal in most jurisdictions. We decided not to call it “insurance” but use the term “risk sharing” instead. Later we learned that avoiding terminology doesn’t help much with regulatory compliance. Also, the fact that the contract does not check for tickets made it look more like gambling than insurance from a regulatory perspective.

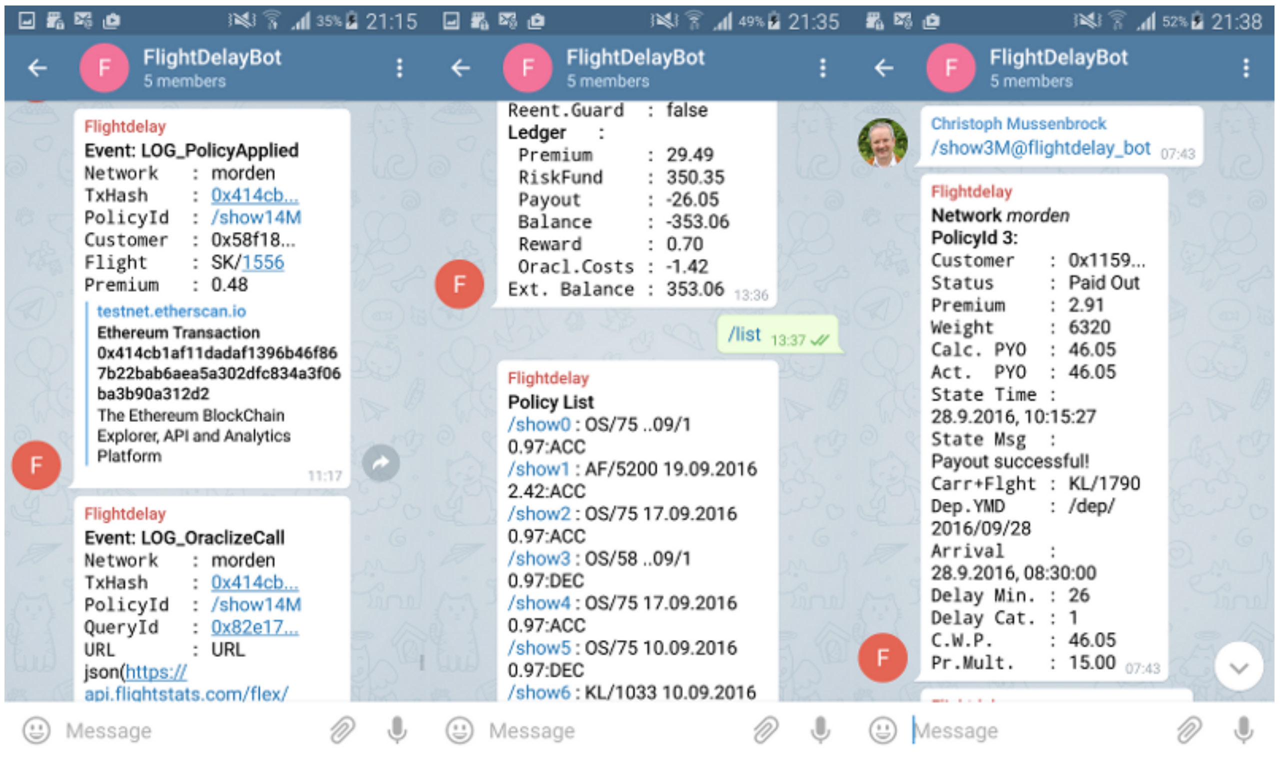
Central control. Of course we want decentralized governance! Yes, please! In the end this is one of the main reasons why we are so excited about blockchain. However, we all know that bugs are inevitable, and bugs in smart contracts tend to have drastic consequences. So some form of emergency control and upgrade paths are necessary, and this is much easier to realize with central control; we called it “maintenance mode”.

Like Christoph Jentzsch put it nicely in his highly recommended talk at devcon2: “stepwise from centralization to decentralization”. It seems many more steps are needed for fully decentralized governance; probably this will not happen before we have formal verification for smart contracts.

Game theory. It took us some iterations and many tests with real users to balance incentives and set the right limits. Incentives are likely to change over time, so it is probably a good idea to keep relevant parameters configurable. Only real users and real money will reveal game theoretical risks.

Monitoring. Good monitoring tools are crucial. At least in the beginning (i.e. the first 100 transactions or so) you want to get notified on every transaction and you also want to make sure you understand exactly what happens in each single one.

We used a Telegram bot to receive log events in real time and to query the status of the contract and policies (screenshots below). This made monitoring the contract very convenient, and we could investigate issues quickly from our mobile phones.



```
Flightdelay
Event: LOG_PolicyApplied
Network    : morden
TxHash     : 0x414cb...
PolicyId   : /show14M
Customer   : 0x58f18...
Flight     : SK/1556
Premium    : 0.48
testnet.etherscan.io
Ethereum Transaction
0x414cb1af11dadaf1396b46f86
7b22bab6aea5a302dfc834a3f06
ba3b90a312d2
The Ethereum BlockChain
Explorer, API and Analytics
Platform
11:17

Flightdelay
Event: LOG_OraclizeCall
Network    : morden
TxHash     : 0x414cb...
PolicyId   : /show14M
QueryId    : 0x82e17...
URL        : URL
json(https://
api.flightstats.com/flex/
```

```
Reent.Guard   : false
Ledger    :
 Premium      : 29.49
 RiskFund     : 350.35
 Payout       : -26.05
 Balance      : -353.06
 Reward       : 0.70
 Oracl.Costs  : -1.42
Ext. Balance  : 353.06 13:36

/list 13:37

Flightdelay
Policy List
/show0 : OS/75 ..09/1
0.97:ACC
/show1 : AF/5200 19.09.2016
2.42:ACC
/show2 : OS/75 17.09.2016
0.97:ACC
/show3 : OS/58 ..09/1
0.97:DEC
/show4 : OS/75 17.09.2016
0.97:ACC
/show5 : OS/75 10.09.2016
0.97:DEC
/show6 : KL/1033 10.09.2016
```

```
Christoph Mussenbrock
/show3M@flightdelay_bot 07:43

Flightdelay
Network morden
PolicyId 3:
Customer     : 0x1159...
Status       : Paid Out
Premium      : 2.91
Weight       : 6320
Calc. PYO    : 46.05
Act.  PYO    : 46.05
State Time   :
28.9.2016, 10:15:27
State Msg    :
Payout successful!
Carr+Flght   : KL/1790
Dep.YMD      : /dep/
2016/09/28
Arrival      :
28.9.2016, 08:30:00
Delay Min.   : 26
Delay Cat.   : 1
C.W.P.       : 46.05
Pr.Mult.     : 15.00 07:43
```

Screenshots of the Telegram bot we used for monitoring the contract

Cross-contract. Interaction with third party contracts means many moving parts and is sometimes really hard to debug. We got tremendous support and amazing reaction times from the oraclize team, thank you, Thomas and Alice!

Time delays. A minimum time of 24 hours (plus flight time) between application and payout was sufficient for us to monitor and balance the maximum risk exposure.

Invariants. We used an internal bookkeeping system to keep track of amounts paid in and out. This made it easy to spot strange behaviors, bugs and leaks.

We also want to thank Nick Johnson for making solidity-stringutils available, and the MetaMask team for their support and great work.

Last but not least we thank our friends and sponsors for their courage to provide financial backing for potentially surprising payouts: Markus Kuhnt from disrupt consulting, and Michael Hoesch, CEO of lifebond.

Some more thoughts on block gas limit, contract complexity, automated deployment, and other technical and development related issues will probably follow in another post.

Next steps

The user interface at fdd.etherisc.com has been updated and a new contract without deadline has been deployed on testnet for you to keep using and testing. Note that all contracts we deployed on main network are now locked down and will not accept new policies anymore.

- Security audits. Security in smart contracts matters most. We are in contact with potential auditors and security experts, if you are interested in conducting or sponsoring an audit, please contact us.
- Regulatory safety. Using smart contracts for real insurance applications needs regulatory approval, and we are keen to explore that further. We are in contact with regulated companies and regulatory bodies in different jurisdictions.
- Partnerships. We have been contacted by many interested parties and are talking to potential partners in order to evolve this into real business applications. Our main focus is to get access to insurance markets and an integration in a flight tickets buying process.

Are you interested in working with us? Join our team at hack.ether.camp, a virtual hackathon starting mid of November, or contact us at contact@etherisc.com