II

MAIN OPEN QUESTIONS & RATIONALE OF SOME EDITS

I think the paper is much improved now, fit for resubmission, without changing your order or intent.

1. EDIT PROCESS

I tried to answer all questions that arose through the review myself, based strictly on what I could identify as your intent. I aimed at reducing your workload as much as possible by avoiding out-of-context inquiries to you, and because we did not really have a cadence going that allowed for a quick interchange of questions. The upside is a nearly ready-made paper for your review now, with relatively little input from you so far. I believe that in all, going through the following at your own pace will save you time and nerves, and avoid misunderstandings.

2. YOUR REVIEW PROCESS

Please read the secondary documents first, starting with the 'laundry list':

File '**Dream -- (0) Viktor's Laundry List (new July 25)**'.

When you get to this doc here, it has its topics grouped for severity. **Red** you must address (re-use of one-time pad), **orange** you could double check or ignore (expiration, updates, termination), **blue** are changes I found unavoidable (indexes, chunks). I am proposing solutions in **green**.

You could then **try** the following for fastest results:

1. Get a quick impression by looking at the Overleaf editor
2. Check the changes to the formulas only
3. Make a copy that accepts all changes
4. Edit that new copy to become what you would prefer

You might find that this is a fast way to arrive at a satisfying new revision to re-submit.

Only if that does not work would I in your place confirm/reject the edit changes step-by-step. There are hundreds of edits but in fairness you probably get through them in a day as well.

5. Check the comments in Overleaf

3. CHANGES AFTER THE REVIEW

I did NOT change the structure, did NOT rearrange any paragraphs, sentences or chapters.

That means, I did not change the build nor the order of the overall text, nor mix sentences around within paragraphs, nor reorder paragraphs within chapters.

I only corrected mistakes and added text and graphics, sometimes a lot, to follow the **reviewers' request** to slow down the pace and make the paper "**easier** to understand for a broader audience," and clearer regarding its "**implications**." Always with a view to not create irritating redundance.

There were a number of smaller errors with variable names and formulas that surfaced now on a deeper parsing. Some indexes were surprisingly hard to fix. I added variables, like **cc** for chunk content, to **simplify**. It will be easy to rename it to a better name, if wanted, wholesale. All rationales for changes that cut deeper are explained below.

I tried to nowhere change meaning, and I am confident I did not. The exceptions, with a lot of back-and-forth are the role of "node 0" and "node n", and eliminating the use of "chunk" for **byte{,4096}**.

I allowed only added text in such a way that it does not become repetitive. This might appear different to you because you know what is coming (that is, you might find some insertions useless). But I tried to carefully establish a less steep build that explains things in easy terms first, while always avoiding antedating the details that come later. I also added an off-ramp at the end (in Conclusion and one extra section before that "Implications"), at the behest of the reviewers and in part using their direct proposals. These added paragraphs likewise zoom out and remain high-level.

Where changes are not corrections, they are **at all times motivated by the reviewer requests**. To make sure that no content is inadvertently repeated I went through the paper end-to-end twice to double check the flow. While the additions certainly reduce the hard-core elegance of the paper, I also think it is a better read now.

You might find differently; I can imagine that you may want to cut out 30-50% of the new text, and I therefore wrote new things in a stand-alone way so that they will not be missed when cut out. You just accelerate the pace again basically, if you do. That is, *I did NOT weave new language into pre-existing paragraphs that would reference back to newly added text*. The new stuff can thus be cut back out without the need to adapt anything else. (Not that I would cut it, else I would have after re-reading.)

I believe there is little to no redundance even regarding the final definition of the dream protocol and the original circular graphic of the dream path (now fig. 4), which could be seen in competition now with texts inserted above and with that many-spheres graphic (now fig. 1) I added before it. But I believe your unchanged definition of the dream path remains the pinnacle that is but easier to understand now for what comes before. The circular depiction (fig. 4) now presents as a clean high-level abstraction vs. the new, more detail-oriented, Swarm-explaining graphic (fig. 1) that shows both Swarm level hops and dream path stops.

One exception are graphics captions, which are more repetitive vs. the main text. They are intentionally somewhat in a track of their own, like a review of the main text. But then graphics are almost always a redundant alternate way to explain the same thing the text already explains. Still, the captions **may be too long now**. I don't know what is expected or how you prefer it. The goal for how

they are done right now is for the reader to NOT have to have a full grasp of the implications (yet), and thus not being able to skip intermediary steps of the explanation, but instead to be taken by the hand and understand the big picture from the graphics and their captions **step-by-step** in a separate, consistent go in parallel to the main text. All to make the paper easier to follow. (The graphics are still always abstractions, leaving some things out that the main text details.

However, the graphics also speak for themselves. You could cut the captions down a lot without making the graphics useless. The variable names in the graphics are in sync with the main text.

Some questions have surfaced during editing, only one open question remains from the reviewer comments, and some things are explained below that I think you will find helpful.

4. DENIABILITY

I think it has to be addressed stronger that A′ could be used to crack C′ and more should still be done about it than what I fixed so far.

I did only patch the text up and added explanations to defang the correct critique of reviewer 1. It is not really convincing yet I find.

The vulnerability might go away, if **both A and C are themselves encrypted**, before being encrypted again by the dream pad. Because breaking the dream pad k by A′ ^ C′ only works if A and C sport the typical patterns of a plain text: spaces and letter frequency are spottable in A′ ^ C′. But if they were both encrypted themselves, there would be no patterns to go by and A′ ^ C′ might not help to break the key.

Of course, that is not a usual example in cryptography because why would you normally encrypt anything with an OTP if you need to double-encrypt it to make it more safe? In this case though, the other, prior encryption would be to protect the *deniability*, not to make unreadable. And the OTP encryption is not for protection but *deletion*. So, there is a separation of concerns that might make that a justifiable pattern.

It would also make the case for addressability stronger I feel.

The asymmetric key for C could be freely shared.

It is not too involved compared to the rest of the system to double-encrypt A and C.

The downside to deniability is, would it look suspicious to encrypt A a second time (with the OTP key) when it is already encrypted? But an answer could be: no, it is simply to make it deletable (ostensibly). And that double encryption is just what the protocol requires. Just like it is for C in reality.

And to make it deniable that A is just a decoy, we could put B up there. Or not, and just say, A is the real thing. But we don't do the effort to create a deniability decoy B. But because the protocol demands it, A is still being double encrypted. Anyone could always claim, to just have uploaded A

the Swarm way, double encrypted, but did not bother to create a B to have better deniability. It's just the protocol that requires things this way. While in reality having created A to deny C.

You could alternatively also argue, and be open about it, that this kind of XOR-pattern-finding hacking (see images shared on Telegram) would be so involved that it does not count as breaking the principle of deletion-by-removing-access. Maybe yes, it's not *completely* reliably access-removed because of that danger that A' could be used to decrypt C'. But the effort to find A' and C', then breaking the encryption … if that is needed to get at the 'deleted' data … that might be safe enough. However, in that way, you are still having a too weak case of deniability itself. No matter what the deletion argument is, you can't deny very well that C' was yours because the XORing reveals that.

Therefore, I would propose to encrypt A and C before encrypting them with the OTP. If that works the way I think that double encryption would make double use of the OTP safe.

It is possible that encrypting **only** C would suffice for A' ^ C' to be useless to hack either A or C. Possibly, then, A' ^ C' would also not reveal patterns. I did not research that.

Regarding double-use of OTP see: https://incoherency.co.uk/blog/stories/otp-key-reuse.html

5. EXPIRATION

There is a claim that a key can be made to be usable only once.

There seem to be no details about that in the paper though.

> *"E expirable – The scheme allows for one-time use in this design, i.e., that the key can only be retrieved once."*
>
> Section 3.2, pg. 5

You might want to revisit that and delete or explain it, at least with one more sentence?

6. TERMINATION

How does a node know when $\mathbf{n}$ is reached, and the dream path ends?

I don't think your list of "termination criteria" catches this. It just presupposes that knowledge.

In order to guarantee the correct termination, the following criteria must be fulfilled:

> *– all buckets of batch b designated by a_i for all $0 < i \leq n$ must be non-empty,*
> *– the length of the prefix shared by the addresses of chunks in the same bucket must be greater*

than the storage depth d of the network,

– the chunk ci of each step 0 ≤ i < n must be sent to the neighbourhood designated by ai = H (ci),

– nodes at each step 0 < i < n of the dream path must both compute the OTP update and push the output pad as a chunk to the network,

I added a bullet point to this list but it likewise does not help to understand when **n** is reached.

n is not part of the metadata, and you are explicating that nothing about the step count can leak.

It cannot be a network constant, as the network can grow and **n** needs to be bigger than **d** to be safe (?). It cannot come from an oracle because the travelling dream pad is not leaking even to the receiving and processing node on what step it is.

It cannot be signaled by the buckets of the batch, i.e., not be implicit in the metadata beta. Which I understand is the only bit of metadata travelling with the dream pad.

I assume that you meant that the receiver at **a** knows that the dream pad is for them and stops the travel. But that would require that every node in the neighborhood knows it is coming and they are not all going to be informed.

If you have an answer maybe it could be made clearer in the text, what it is.

7. DELETION BY UPDATE

The following might be a misunderstanding on my part overlooking a 'trick' based on proximity order. Obviously, I don't find where I am wrong. But I did find that you changed how **χ( )** is employed only recently (April). Maybe you overlooked the following.

Per the paper, an update chunk **u** is posted by the uploader to 'delete,' i.e., to undo a working dream path.

For this **u**, to be selected by the delta function for XORing with an incoming chunk **c** on the dream path, it must have the smallest XOR distance **χ(c,u)** compared to all other chunks' **χ(c,other)** under batch **b**. The distance is **between the address of the incoming chunk and the respective reserve chunk**.

That means that to construct a fitting **u** that will derail the dream path to effectively delete **C,** one has to know

1. against **what** incoming chunk **c** the XORing distance check is going to be, to get it right that one gets the lowest distance in place with the update **u** vs. the incoming **c**.

But there is no good way of knowing what the **a** of the incoming **c** on the dream path will be. It is only certain that it is in the neighborhood.

If there is a property of **PO** distance used that you have in mind that I didn't catch that is implicit in the relationship of batches, buckets and neighborhoods, and that makes it trivial to

generate a **u** with a lower XOR distance to ANY incoming **c**, you might want to make it explicit in the paper.

The same if I have failed to understand what the address **a** is (not the incoming chunk's address?). But I think you possibly only recently changed against what XOR distance is to be minimized, should I remember an April version of a paper correctly.

2. One also has to know what distance all other chunks in the bucket will have to mine a smaller one. This is relative to the incoming chunk and cannot be predetermined.

So you would need to know all addresses in the batch (bucket ?) stored at the neighborhood under **b**.

3. I understand that you mean that this is simply tried out but mining by sending requests to a network. This sounds like a lot of traffic?

4. Else, this requires emulating the dream path or being able to find out what $\mathbf{c_i}$ will be at a given node (or rather, certain neighborhood) where one positions the update.

Maybe precisely that knowledge could per dream protocol definition be shipped back to the uploader when the dream pad is first created. It is the layers of the onion skin, so to say, the evolution steps of **k**.

If this is not solved, I'd propose not using the **a** of the incoming chunk, but maybe the bucket's or the neighborhood's? So the task is only to minimize distance to the bucket pivot compared to all other chunks already uploaded.

The second paragraph of the new section "Implementation Constraints" (added at the request of reviewer 2 – the one who understood the paper – explores the idea of caching the information needed for a local simulation–local mining–of seed **g** and updates **u**. It can be cut in its entirety if off.

8. INDEXING

I belabored the following three points that are interconnected.

The use of **i** for indexing nodes, chunks, addresses and payloads was inconsistent in the formulas but I could not get it right for a while.

I know why now, and I think I found the best way and corrected everything to that.

But if you don't like it I am happy to change the required places to a different logic, just let me know.

NODE ITERATOR

It remained unreconcilable to me how you used the **i** index in connection with dream chunk and nodes. It seemed different in different paragraphs and formulas.

Of course, that looks like a non-issue but it had led to multiple off-by-one errors.

I eventually settled for the premise that you most likely want **pi, ci, and ai to be created at node i.**

That in turn means that the incoming chunk is **ci-1**, arriving at a_{i-1}, so the argument to **Δ** is p_{i-1}! The more un-elegant downside being that a_{i-1} is the address of **node i**.

You sometimes had it all shifted up by one vs. the node index (so a_i arrives at **node i**), but sometimes not. The main Shelling points are, either does **node i** create c_i, or **node i** has a_i as address.

Only one can be, unless you define the address of a chunk to be one ahead of the chunk index itself, $H(c_i) = a_{i+1}$. But then you have as dream chunk $<a_{i-1}, m \bullet p_i>$, which is too confusing I think.

What played directly into this topic and made it more confusing is the question – seemingly merely a matter of taste – what Swarm node should be **node 0**:

NODE ZERO

After much cross-examination, it remained unclear how much agency is projected into the role called **uploader U**, as opposed to what the **local client** does for the uploader – and whether that local client should be seen as **node 0**. A view you seemed not to have.

Eventually I decided, it should–veering from the rule to not change your decisions–for the following considerations:

If the **uploader**, as was written, *"expands* **g** *to* **p***, by using* **G[4K]***, wraps it into* **c** *and sends it …"* – surely, then he is using a program for that? Programmed by the Swarm team – and that would probably just be a local client. And not a different auxiliary script. So it would be F_0 rather than the uploader **U** himself.

The question really is, who is **node 0**, who is F_0, are they the same, and what degree of pre-processing of data should F_0 realistically except – I think none. It should just receive the raw data from **U**.

Which is what you seemed to indicate in the caption for fig. 1, which clearly had it that uploader **U** *"posts the chunk to local client* F_0*."*

So, if with 'chunk' in this case you meant the raw data, **byte{,4096}** that would mean the **uploader** did NOT pre-process the data. It's the **local node** F_0 that is wrapping the payload into a full-fledged chunk and calculates its address **a** in **S**.

But in your circular depiction of the dream protocol depiction (ex fig. 2, now fig. 4), the first node, N_0 was to be found **by G[4K](g)** (I am guessing again inspired by your understandable intuition that the first address a_0, created by **G[4K](g)**, should lead to **node 0**) and in this case the uploader **U** seems to be doing the calculation **G[4K](g)** to find N_0 himself. Or maybe F_0 is doing that for him.

But if the *Uploader* calculates p_0, c_0 and a_0 **= chunk 0** then this **chunk 0 =** $<a_0, m \bullet p_0>$ **arrives** at **node 0**, which is then creating **chunk 1 =** $<a_1, m \bullet p_1>$. Which I think you would agree is a confusing phase shift. **Node i** always thus would create **chunk i+1 =** $<a_{i+1}, m \bullet p_{i+1}>$**.**

And again, this *does* make sense for $\mathbf{a_{i+1}}$. Because that is the address of the next node. But otherwise, I figured that this is not what you will have wanted and it looked confusing in the formulas. It would be better if **node 0** was producing $\mathbf{chunk\ 0 = <a_0, m \bullet p_0>}$.

Because of the special case of $\mathbf{p_0}$ this **node 0** should then be = $\mathbf{F_0}$ and it should be executing $\mathbf{G[4K](g)}$ for the uploader instead of the usual upgrade function $\mathbf{\Delta}$.

In other words, what $\mathbf{F_0}$ does for the user makes $\mathbf{F_0}$ the first program of the dream protocol-aspect of the network to be involved, that is, it should be **node 0**, which does not do the regular update function, but $\mathbf{G[4K](g)}$.

If not, you have $\mathbf{F0, F1, \ldots S = node_0}$, a number of hops in the network, before you even reach **node 0**, which then creates $\mathbf{p_1}$ etc. and it remains unclear who is responsible for the first step in the dream path protocol, to produce $\mathbf{p_0}$ from $\mathbf{g}$.

I hope you find this convincing. I am spelling it out in detail because the change is somewhat pervasive. A change of the iterator logic figures in formulas, texts, graphics and descriptions.

But if you want a different phase of $\mathbf{i}$, I am happy to update the doc to reflect that.

LAST NODE

I proposed edits in a few places in the paper to have node n **also use the update function** Δ on the incoming dream chunk, so as to calculate dream pad k from it, rather than just "extracting" it from the incoming chunk, as it was. Because if the last node does not use $\mathbf{\Delta}$, why even send the dream chunk there. Just the 'extraction' and backwarding could just as well be done by the node before, which then does not actually extract but calculates k and backwards it immediately instead of forwarding it to yet another node to do that.

Am I missing that it would be more secure that way for some reason? Because it's part of the addressability? I think that doesn't change the picture. I can edit it back if you prefer.

For now, I changed 2 sentences so that the last node, at address **a**, **also** uses the update function to for a last time modify the dream pad before backwarding it to the uploader or grantee.

This is also connected to the question of what **node 0** does. You describe the dream pad as "the result of $\boldsymbol{n}$ applications of the OTP update function." Therefore, if **node 0** does $\mathbf{G[4K](g)}$ (as I describe above why that makes sense) and **node 0** is where the hopping starts, $\mathbf{n}$ hops must end with **node n** and a last application of $\mathbf{\Delta}$ there.

However, I am happy to change this back, if I am overlooking something there, why the last node should not apply $\mathbf{\Delta}$.

9. CHUNKS

I propose a major clarification, namely, **what a chunk is**. This may sound non-sensical to you, but it was a major distraction when first reading the paper.

I edited these changes in (in track-edit mode of course, like everything else).

I tried to do it *with as little changes as possible* and eventually turned to the definition of 'chunk' in the Swarm book to be sure I was on the safe side. It turns out to be the same like what was spelled out more or less explicitly in the paper in some formulas. Although also in the book you use multiple meanings in the main text body.

C C

I use cc to mark the 'chunk *content*.'

cc is not super pretty but it also stands out and that's not wrong for this 'dual' meaning: not the entire chunk, not just the payload either.

I think no neutral reader not versed in Swarm would find it obvious or easy to remember that the address is NOT officially part of the metadata (because really, it is, isn't it. Just by necessity it can't be part of what is hashed.)

If **ɣ** was free, I'd use it btw., instead of for blockchain height. That looks a good fit, with **c** for 'chunk'.

C H U N K S

My take is that you grew so accustomed to the word 'chunk' that you just don't realize anymore that you are using it in four different meanings in the paper (**byte{,4096}**, **byte{, 4104}=m•p**, **<a, m•p>**, **<H(m•p), m•p>** the latter with some rights as special case). Which is confusing. Let me explain:

I think you did realize that yourself and the paper in some places made the clarifying distinction between "chunk message" and "content chunk," which would be helpful if the distinction was used consistently. But it would still contradict the definition elsewhere in the paper that clearly define a 'chunk' as the 'chunk message,' so to say.

The paper, however, led in with *"We will base our exploration on chunks, the fundamental **fixed-size** storage units"*, *"e.g., **chunk-sized = 4K**"*, *"a dream is a **chunk-sized** one-time pad"*, and *"When content is uploaded to Swarm, the local client splits the data **into 4-kilobyte chunks** and pushes each chunk to the network."* In all these cases chunk means a <=4K fragment.

One read from then on with the idea that a chunk is 4K raw data. While you will have rightly assumed that details are best saved for later, the confusion between the content of a chunk and what the richly structured Swarm chunk really is never went away in what followed, if one tried to follow the details of the formulas and not just the big picture. You went on to use 'chunk' and 'chunk content' interchangeable, then made a distinction between 'content chunk' and 'chunk message'; but it never became quite clear what the pure 4K content payload was called and what meta data is added for hashing – and if the metadata is part of the content in that case, as seen from the address hash ...

Again, for you that will all be so obvious that you might not see a problem. For a reader of the Swarm book, for sure, too. And the graphics from the Swarm book would have been an easy way to clarify. I saw that the one for single-owner chunks is in the Overleaf folder. The one for CAC would have been useful instead.

That it was most difficult to understand is not a theory, the blur was my experience for days. A reason is that one does NOT, as a reader, pay overly much attention to understand all basics in the first attempt. One tries to fill in the blanks on the go, one after the other and the chunk issue really gets in the way because one is led to believe *to have misunderstood* something whenever 'chunk' is used again in a surprising way and things don't quite add up.

Eventually I found that the lack of clarity about what a chunk is leaked into other definitions, e.g. $\mathbf{Delta : Chunks \longrightarrow Chunks}$, when it is really, quite certainly $\mathbf{Delta : Chunks \longrightarrow Data}$. It only made sense that on the logic level you seemed to have made an attempt to fix the ambiguity yourself (using 'content chunk' and 'chunk message'), just did not quite complete that approach.

When the request from reviewers came back to try to make the paper easier to understand I first tried to make mentions of 'chunk' non-ambiguous by adding diacritical math terms like $\mathbf{byte\{,4104\}}$, $\mathbf{<a, d>}$ as respectively appropriate, right after the word 'chunk' everywhere it occurred. But it turned out that adding the clarification in math terms did not work because in some instances, the text was only correct when 'chunk' meant two different things at once.

At some point I suspected that you felt that $\mathbf{<a, d>}$ is *redundant* with $\mathbf{<d>}$ for CAC because $\mathbf{a=H(d)}$, and so $\mathbf{<a, d>}$ is 'equal enough' to $\mathbf{<d>}$ in meaning that they can both be called the same. But because a chunk is really $\mathbf{<a, m+d>}$, this does not work, at the latest, when it's time for hashing a chunk. So I dropped that, too.

I then tried if it could all be simplified by 'chunk' to always mean $\mathbf{<a, m•p>}$, with the allowed special case of $\mathbf{<H(m•p), m•p>}$. When that was still impossible to be made consistent, I turned to the Swarm book to make sure that any wider-reaching unification I would propose was in sync with the use of the word in Swarm's wider context.

Basically, the decision had to be made to ether use 'chunk' colloquially as 4K of data–which is from user perspective all that ever matters–or for 'chunk' to mean the essential Swarm protocol unit that includes overhead. My proposal is to drop the colloquial use. And to avoid it in all instances but treat 'chunk' as a well-defined term.

I think you might not like that as it reduces elegance. It undercuts but a mirage of simplicity.

As well, the Swarm book, and the defining formula in the paper clarify explicitly, that **a chunk includes an address**. This is also the opinion of Wikipedia: a chunk has some overhead.

The Swarm book unambiguously has

> *"[Chunks] are an **association of an address with content** ... their content limited and roughly uniform in size"* (pg. 53 / 73 of the pdf)

As a neutral source, Wikipedia has an entry for the meaning of chunk in IT:

> *"A **chunk** is a fragment of information which is used in many multimedia file formats ...*

*Each **chunk contains a header** which indicates some parameters (e.g. the type of chunk, comments, size etc.). **Following the header is a variable area containing data**, which is decoded by the program from the parameters in the header.*

Chunks may also be fragments of information which are downloaded or managed by peer-to-peer programs.

In Distributed computing, a chunk is a set of data which is sent to a processor or one of the parts of a computer for processing."

That clearly makes a chunk a bundle of content AND meta data. Interestingly, the third sentence is as ambiguous are your more liberal use of the word.

So I used content+address as the definition of a Swarm chunk and moderated all other uses of 'chunk' to smoothly mesh with the more precise articulation.

I decided to borrow the word *fragment* to replace your colloquial use of "chunk" in the very first instance. I think this will be helpful for the reader to avoid associating chunk with the wrong meaning at the very first use of the word.

I therefore unified:

- a 'chunk' is content and address
- 'content' is all that is not the address, like you show it in the Swarm book's graphics
- Raw 4K are 'payload', 'data', or 'fragment', whatever reads best (but NOT 'chunk')

This reduces the momentum of the text a bit (just always writing 'chunk' made for brasher reading) but is clearer.

One could define 'chunk' differently, explicitly ambiguously, why not, to be able to use it the way you did. (And if I were to decide, I would go for chunk = 4K data instead!). But it should better be consistent with the explicit definitions of both the Swarm book and the paper.

I am making a guess that part of the blur surrounding the use of the word 'chunk' – especially some unclarity about the meta data – came from that you wanted to preserve a place for the single-owner chunks of Swarm – that are not content-addressed – so that they could *also* fit the mold the paper provides. Even while they are not mentioned in the paper. But it is obscure to the reader of the paper why some aspects of chunks would be kept vague, why there is even a distinction between 'chunk' and CAC! It may also not be necessary to keep the door open for single-owner chunks because the CACs are included in them.

This all as explanation why I am proposing the changes, and not as an excuse to make the changes any bigger than needed. But as a reader it was a major irritation that deserved addressing, really even without the reviewers very general comments.

H (X)

A problem remained that you also use c_i sometimes for a Swarm chunk including address, or for the payload plus metadata without the address. That made H(c) ambiguous. One gets what you mean, and it is not a real problem. Some details are implementation choices at any rate.

But I think it was not correct. So I made the definition of H() explicit and to have cases for different type arguments. This is defining H() explicitly as H(chunk) = H(payload(chunk)) essentially.

PAD XORING

You seem to repeatedly indicate that the ***entire*** appropriate reserve chunk is XORed with the **payload** of the incoming chunk. I think that's wrong but did not edit that because I am not certain if I am overlooking something as you make the clear distinction with the incoming chunk (that the pad is only the payload, not more) but not the reserve chunk. I wasn't sure if you possibly really mean it.

Even after understanding that you used 'chunk' for both meanings, I was not sure if there was a special circumstance here, because I am not sure I am overlooking something with the 'deleting-by-update' procedure, where an update $\mathbf{u}$ should deplace the former highest-PO chunk in the reserve under b.

I think only the *payload* of the reserve chunk can be XORed with the incoming chunk, otherwise you have too many bits. But maybe there is a security issue with using just the payload, or from including the address that you are aware of and imply with your choice. I also run into trouble making the graphics right where I tried to be clear about the difference between chunk and payload. I am suspecting you actually meant XORing with the payload. But as its written I keep with the notion how it reads: the entire chunk.

Let me know if that should be changed. It would mean to touch up some formulas and graphics.

SPAN

Apropos, what eludes me still is whether you just dropped the $\mathbf{span}$ for the paper's definition of a what a chunk is? To have the pad $\mathbf{k}$ fit the content and then just assume it's being padded always?

But what is the advantage of getting **incompatible with the content-addressable chunks** of the Swarm book that have a span, instead of a batch part as metadata?

Why do you NOT need a span here? Is it to make the pads camouflaged by the normal content-addressable chunks?

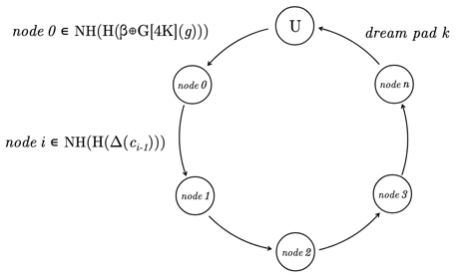
10. RESERVE Q

I named the reserve $\mathbf{Q}$ because that letter was still free. $\mathbf{R}$ and $\mathbf{S}$ not.

I named $\mathbf{q}$ the chunk selected from the reserve by comparison of $\mathbf{PO}$, which is easier to remember than $\boldsymbol{\chi}$ with overline. I found $\boldsymbol{\chi}$ overline irritating as it is such a different type of thing from the XOR distance function $\boldsymbol{\chi}$, as well as unusual for the name of a chunk.

$\mathbf{q}$ being element of $\mathbf{Q}$ appears easier to remember. It rhymes with the payload $\mathbf{p}$ of the incoming chunk, too, that it is XORed with.

11. DREAM PROTOCOL DIAGRAM



I made a new version of fig. 4 (ex. fig. 2, the circular depiction of the protocol U ⟶ N ... ⟶ U).

The new version looks like the old version, just replaced "**N**" with "**node**" to make it consistent with the notation in the rest of the paper, and I corrected the formulas.

You don't explain the function **NH( )** used in the diagram. I added a brief mention of it to the caption, assuming that it means the set of the nodes in one neighborhood, defined by an address that falls into it.

But regarding the original diagram, both the values of **G[4K]** and **Δ** (which are parameters to **NH( )** there) are **byte{,4096}**, not addresses. So, **NH( )** would have to include a hashing of its argument to convert them to an address. I inserted a **H( )** call instead. In the old diagram, **Δ** was also wrongly receiving two arguments, **b** and **c**.

(Note that you defined the value of **Δ** as **Chunks** type in its formula but as mentioned above, I corrected that as it is not how you use **Δ** in the text. It is instead clearly returning payload, i.e., **byte{,4096}**.)

12. BATCH ID

The batch b has its own ID. Does that ID interplay in any way with the chunk and node addresses, the bucket slot spaces?

I understand that this is not so. But in case I am missing something, you could **clarify that in one sentence** maybe.

13. BUCKETS

Even after some reading in the Swarm book I confess I did not develop a firm grasp of what buckets and slots are. I get the idea but it remains round-about. Since you are mentioning buckets in the paper but don't explain them, this remains unclear.

I left it at that because it seems not to add anything essential – maybe just don't mention the buckets or explain them briefly?

I suppose **I would cut their mention** as they don't add to the concept explained in the paper but are a Swarm-internal aspect.

14. ADDRESSABILITY

I feel addressability is an optional element that fits to Access Control and the 'A' in **dream** but is not central.

However, I did not change anything, nor call it 'optional.' Especially, because I was not sure if it plays a role in termination?

But I also don't feature addressability much in the new texts I added, so in comparison it gets even less attention than before.

No action required if you are ok with that.

Polishing the paper more might mean **adding explanation about how addressability is optional** (if it is), or why it is essential, if it is.

15. OVERLAY ADDRESS

Why is it called "overlay address"?

> *The protocol is responsible for transferring newly uploaded chunks to the neighbourhood $\mathcal{S}$ where they belong based on **overlay address** proximity to its content address.*

I am guessing it's to distinguish it from the IP addresses that the peers know of each other?

Is the term "overlay address" very well known, just I didn't know it?

Or should the word be **struck because it might be confusing**?