Matthew's thoughts:

First step should be to check everyone's aligned on the problem, which from my pov is:

- Provide a replacement to GCM/FCM/APNS for notifying clients when they have notifications (and only notifications)

There are some questions around this though:

- Should the notification be 1-bit of data to wake up the client to ask it to call the /sync CS API, or should it contain more data?
    - Currently we send a single wakeup bit via APNS & GCM because we don't trust Apple & Google with message data or metadata
        - However, if we're running or selecting our own push server, we don't care so much about that level of paranoia.
    - Also, we send a single wakeup bit because if we sent an E2E payload we sometimes have to wake up the client anyway in order to sync with the server to establish the E2E state required to decrypt it and display the notification.
        - However, we could improve efficiency for the common case by sending the E2E payload and having the client only wake up and call /sync if it needs to.
    - => therefore, we should consider sending event payloads to clients again rather than just a wakeup bit.
- Should this be an entirely different protocol to Matrix (optimised for push), or should it just be an alternative transport for Matrix itself (which we happen to use for syncing notifications)? For an example of an alternative Matrix transport (albeit not one optimised for push), take a look at the WebSocket proposal at https://github.com/matrix-org/matrix-doc/blob/master/drafts/websockets.rst
    - The advantage of being an entirely different protocol optimised for push is that you don't need to worry about it being a subset of the Matrix CS API, which simplifies the solution a bit, and can potentially directly reuse a simpler existing protocol like MQTT.
        - Matrix's CS API doesn't currently have a filter which says "only send me events which are push notifications", although this would be quite easy to add. It would be critical to add, otherwise we'd waste bandwidth on irrelevant Matrix traffic and activity.
    - The advantages of it being an alternative transport for Matrix itself are:
        - You get E2E encryption for the push notifs "for free".
        - We wouldn't need to have a hybrid solution where the push client wakes up the matrix client in order to ask it to /sync, as the push would go straight to the Matrix client - and the matrix client could immediately pull in whatever other info was needed to decrypt and display the message as a notification.
        - The push client would effectively be a Matrix client available to the whole OS, and could also (in future!) supply other Matrix capabilities to the OS - e.g. managing E2E encryption state; or even storing

searchable logs of local conversation history; or providing a simple OS API for apps to send/receive messages without having to independently talk to Matrix servers.
        - Anyone who uses the protocol for push automagically gets all the benefits of Matrix too
        - We can piggyback on the existing network of deployed matrix homeservers, rather than requiring folks deploy and maintain yet another server.
        - We could then use it as a general purpose network-efficient transport for Matrix across all clients, and not just for Push.
    - we'd need to add a 'only send me push notifications' filter to the Matrix CS API, so that when using the transport purely for push, we don't waste bandwidth with anything other than notification events. This would be very straightforward though - a simple extension to the existing Filter API.
    - => my preference would be to first investigate whether it's possible to build this as an alternative Matrix transport, and only fall back to a push-only transport if this proves too hard.

Then, next step is probably to research various different transports to see what's actually any good in terms of bandwidth, battery usage, latency, and resilience to unreliable networking (i.e. constantly switching between wifi & GSM or dropping in & out of GSM). It's **critical** to consider the transport quality quantitatively in order to make an informed comparison. I also strongly suggest entirely decoupling the question of the encoding used for the data from the transport used to transport that data.

- How many round-trips (and bytes) does it take to set up the transport?
- How many round-trips (and bytes) does it take to send a notification?
- What is the average and spread of latency it takes to send a notification (ignoring network RTT)? (e.g. is it an average of 30s with std deviation of 15s (as you might see if it were polling every 30s), or is it negligible (if it's a long poll or event stream)?
- How much bandwidth does it use to keep the connection established to the server?
- Does it take into account the energy usage of the physical layer? This is obviously easier said than done, but some protocols (specifically GCM and APNS) *do* try to get this right.
    - For instance, Android phones on HSDPA typically fire up their radio for 2-3s at a time (at least last time I checked, around the Galaxy S4 era), and the radio being enabled and transmitting is by far the largest source of battery use (next to the screen). So you want to avoid doing any background task which uses the radio for more than 4s at a time, and to batch up as much as possible within that window. And if the radio is already running due to another app using data, one should always try to take advantage of it to try to check for data, and so avoid being the one responsible for firing up the radio to check.
    - Similarly, cell towers have configurable timeouts they use when pushing data to phones at different priorities, and I believe it might be possible to tune your network stack to match the timeouts in order to be maximally network

efficient. I can't remember the details, but apparently GCM does this; need to research how that happens.

The sort of transports and encodings worth looking at (off my top of my head) are:

Transports:

- APNS and GCM/FCM. Given the goal here is to replace them, we should first understand how they work. Hopefully someone else has done most of the work here.
- MQTT
- MQTT-SN?
- COAP
- HTTP/2 with server push
- Probably **not** WebSockets, which is obsolete as of HTTP/2
- I suggest we deliberately do **not** try to create a whole new TCP/UDP/QUIC transport, as it's almost certainly reinventing the wheel that one of the transports above has fixed already, and it's Hard. That said, it might be worth looking a bit at whether QUIC to see if they've thought about this.

Next step: worry about encodings. These are less important than the transport given they are layered on top, although given some transports only support certain encodings (e.g. COAP+CBOR) it all becomes a bit interdependent.

Again, this *has* to be quantified otherwise you're running blind.

- How much CPU does it take to receive 1000 matrix events?
- How much bandwidth does 1000 matrix events consume?
- Does it compress well (assuming it's not already compressed)?
- Does it factor out commonality between consecutive events? (like HTTP/2 avoids resending the same headers in each response if they haven't changed)

Encodings:

- protobufs? (e.g. HTTP/2 + protobuf, aka gRPC)
- CBOR? (e.g. COAP + CBOR)
- cap'n proto?
- Thrift?
- JSON-B? (I know Phil Hallam-Baker, who invented JSON-B, so could ask him questions directly if needed)
- some other binary JSON representation? BSON? BJSON? UBSON?
- MsgPack?
- I suggest we deliberately do **not** try to invent our own encoding and reinvent the wheel.

https://chadaustin.me/2017/06/json-never-dies-an-efficient-queryable-binary-encoding/ looks like an interesting set of prior work on this.

Suggested Plan From Dave

- Compare efficiency of transports & encodings in Matthew's list above (not necessarily all of them - some might be ruled out for other reasons) (see Matthew's bullets in choosing transports).
- Pick a transport
- Implement proxy that syncs from homeserver over HTTP client/server API and sends events down to the client in the more efficient transport. This will introduce an extra step on the path the notification takes, but should be easier than adding a transport to synapse itself and will also be re-usable for Dendrite.
- Do a spec proposal for, and implement a 'notify' filter in synapse which filters the stream to only events that would be sent via push.
- Implement receiver on Android