HYPELABS

The Hype SDK: A Technical Overview

Table of contents

01

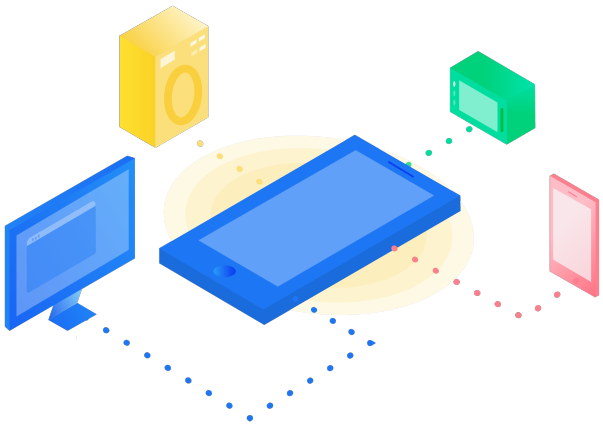
Introduction

Mankind has seen a great deal of innovation since the dawn of the Internet in the early 60s, with the invention of **packet switching** and the early days of the ARPANET[1]. Back then no one had even begun to grasp the technological, social, economical, and cultural impact that this technology would bring to the world. Only in the 90s would that become apparent, with the **World Wide Web** being made massively available to the average household and trigger the starting shot for the **Information Age**. Since then, the **Web** has seen tremendous growth, reaching its first billion users in less than a decade and nearly half the world's population to this day[2], only a little over 25 years after its birth.

But nearly half the world's population being connected means there's another half that is not. At the same time, the Internet's growth is already showing signs of slowing down[3]. The industry is trying to revert this downwards trend by thinking out of the box about reachability, such as Google's Project Loon[4], or Facebook's internet.org[5], ultimately creating a shift on how we are bringing connectivity to the world. Communities don't wait either, and instead growth trends are being seen in community projects such as guifi.net[6] in Spain, Ninux[7] in Italy, and PeoplesOpen.net[8] in California.

The main motivation for this could arguably be the cost of infrastructure. Having reached over US$40 billion of worldwide spendings on infrastructure in 2017[9], the industry is now looking at smarter and cheaper alternatives of achieving the same thing. Open and community projects, on the other hand, are more concerned with bringing connectivity to those disadvantaged, or escaping governmental control when political conditions are less favorable. This is often seen in countries such as China[10] and Cuba[11], where political adversity hampers connectivity.

But it gets worse. Cisco defined IoT to be the moment when the number of devices exceeds the number of people on the planet[12]. Assuming this definition to be correct, claims that IoT is on its way[13] don't make sense anymore, as it already happened, sometime in 2008. Still, that's not what the fuss is all about. IoT brought the promise of a new world of possibilities: all things connected[14]. Lamps, heart monitors, smart watches, cars, clothes, refrigerators, people. **Things**. Yet, industry leaders forecast over 20[15] or even 50[12] billion devices by 2020. With such mind-blowing numbers, many started wondering whether the infrastructure will be ready[16].



"IoT is simply the point in time when more "things or objects" were connected to the Internet than people."

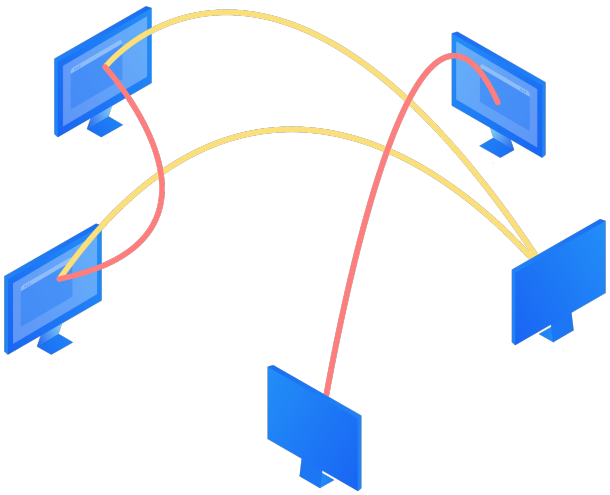
Dave Evans (Cisco)

Yet all of this may be seen as an exacerbation of the scale of the problem. As telecom operators worry about spectrum[17], 5G[18], and the costs of infrastructure deployment for the IoT promise, we still face connectivity issues on a daily basis. This is especially true at gatherings with high counts of people, such as conferences[19], festivals, and sports events, plus the obvious, considering rural areas, or places with blocked signal and underground facilities, like, say, elevators, subways[20], and parking lots. Providing connectivity on many of these scenarios may prove difficult, given that the length of Ethernet cables is limited to 100m (328 feet)[21] and Wi-Fi's range[22] not being far from that. All this means that the problem is not just related with coverage, but also with scale of deployment and environmental factors.

Infrastructure alone doesn't cut it and the industry doesn't seem to let it go unnoticed. Consider, for example, software solutions such as the 3GPP[23] collaboration tackling new standards, like LTE-M and NB-LTE-M[24]. Non-profit organizations realized this too, including ISO's MQTT[25], IETF's 6LowPAN[26], LoRa[27], and other collaborative projects such as IoTivity[28] and Open Mesh[29]. A big problem also means a big market opportunity, and startup ecosystems also responded at the hands of OpenGarden[30], Uepaa[31], Bridgefy[32], and the like. As we realize that software can be a cheap alternative, we look into our pockets to find this extra bit of infrastructure, one that we might have been overlooking so far. This observation introduces a different paradigm of connectivity, one in which the devices are the infrastructure itself.

"As we realize that software can be a cheap alternative, we look into our pockets to find this extra bit of infrastructure, one that we might have been overlooking so far."

Mesh networks contrast with standard networks in topology. Instead of relying on fixed infrastructure, devices are capable of relaying data for each other and can operate as routers. When two devices are communicating, intermediary proxies forward content for them. Without a centralized entity managing the network, such as an access point, the network is not sensitive to single point of failure. It's notable that the medium is variable; that is, mesh networks can be wired or wireless, over Wi-Fi, Bluetooth, or anything else, even combined. Such networks can also be static or dynamic, depending on whether the nodes are mobile or not. But mesh networks are not meant for replacing existing frameworks, but rather to complement them. In fact, when combined with stationary infrastructure, they form hybrid networks, often used for last mile connectivity[33].



"Without a centralized entity managing the network, such as an access point, the network is not sensitive to single point of failure."

Mesh networking research dates far back to the 50s. Back then the United States military was involved in a project called Sound Surveillance System (SOSUS)[34], with the intent of tracking Soviet submarines in the Atlantic and Pacific oceans. But only in the early 80s did we see the same concept applied as we do today, at the hands of the Defense Advanced Research Project Agency (DARPA)[35]. DARPA started the Distributed Sensor Network (DSN)[36] program in 1980 with the intent of exploring the concept of Wireless Sensor Networks (WSN). It was soon after that that the academia took an interest, notably through partnerships with universities such as the Carnegie Mellon and the MIT[37].

At this time the use cases for mesh networks were mostly military, academic, and governmental. As researchers were absorbed by the industry, mesh networks started being applied in other fields, such as industrial automation or power distribution. There has also been great market demand for mesh technology throughout the decades, but hardware and deployment costs, along with other factors such as power consumption and scalability, have hampered this shift for long[38]. But Moore's law brought more computational power to our pockets than NASA had when they landed a man on the moon[39]. These devices are more powerful, last longer, and natively support wide varieties of radio technologies, such as Wi-Fi or Bluetooth. And the best thing is: we already own them.

Hype is an interoperable cross-platform communications SDK with support for multiple types of communication technologies, such as Bluetooth, Wi-Fi, or the Internet. Connected devices form an ad hoc network that allows messages to hop from device to device — known as **mesh networking** — further enhancing range and deliverability. The framework optimizes delivery by segmenting messages into smaller units and dis-

tributes them through all available transports simultaneously — called **transport multiplexing** — using a probabilistic distribution algorithm that bases its decisions on real-time network profiling metrics. All data is securely encrypted, guarded from eavesdropping, even when hopping between other Hype devices. All this complexity is hidden behind an abstraction layer that provides developers a simple and easy to use API, while only requiring minimal setup.

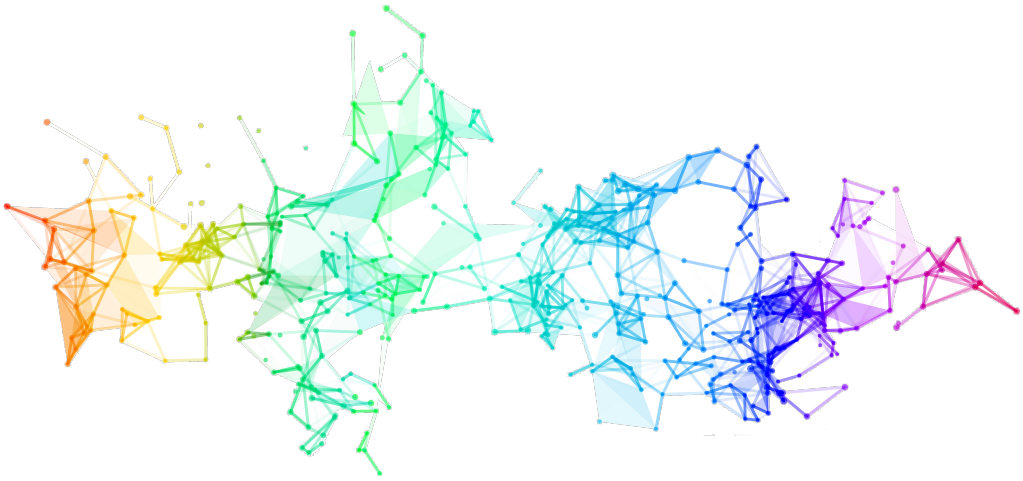
The use cases for such technology are immense. Ranging from everyday applications for smart homes and cities, health care, video games, and social networking, to situations of occasion, such as natural disasters, travelling, conferences, and festivals. The wide variety of applications is further extended if concern is given to market segments exploring devices other than smart phones, such as wearables, sensor networks, home appliances, and so on. This makes Hype the first SDK of its kind for the Internet of Things. With such a wide variety of supported devices and transports, this technology is taking the next step in connectivity, creating secure and affordable solutions for off-the-grid connectivity, network offloading, state of the art Internet reachability, and next generation ambient proximity.

"**This makes Hype the first SDK of its kind for the Internet of Things. With such a wide variety of supported devices and transports, this technology is taking the next step in connectivity.**"

This document provides a not-too-technical overview of mesh networking concepts and specifically of the Hype SDK. This content is provided with the intent of clarifying the limitations and challenges of mesh networking, including how it works and concerns of scale and security. After general considerations are presented, the document goes over specificities of the Hype SDK, by formulating inner workings, requirements, and limitations of the software. This document is meant for everyone with interest in knowing more about the Hype SDK, from a technical point of view. Non-technical readers should still be capable of grasping most of the content, although it was written with individuals with a technical background in mind.

02

The Hype SDK

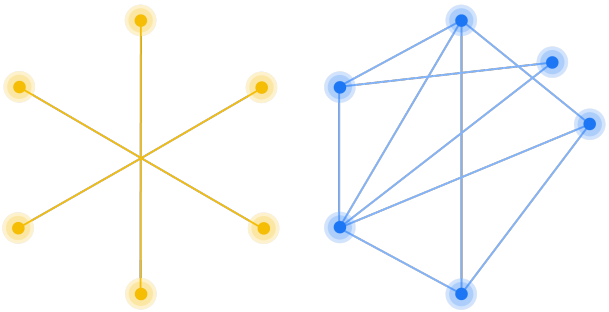


The Hype SDK is a cross-platform mesh networking technology, created with the intent of improving connectivity on all kinds of devices. It works by connecting them together, forming a local network, using existing connectivity technologies, such as Bluetooth or Wi-Fi. This network enables content to hop between the devices, until it reaches a destination or an Internet exit point, improving range and deliverability. The content is protected with end-to-end encryption, meaning that only those participating on a conversation are capable of understanding what is being exchanged. This paradigm eliminates the need for infrastructure, while at the same time taking advantage of it when available. By being agnostic to the type of data that it transports, Hype enables content of any kind of media, including text, pictures, or video. This chapter goes over the concepts and features used by the SDK. Users of the SDK need not understand these at length, or at all, but this chapter was created with the intent of providing a deeper analysis of how the SDK works. Most of the discussions require a technical background, therefore this text is not meant for non-technical people. The following sections introduce features in no particular order.

THE HYPE SDK

Mesh Networking

The concept of **mesh networking** is probably better understood by comparison with other well-known networking topologies. Consider a Local Area Network (LAN) where devices are connected to a central access point. This is called a star topology, and is unarguably the most common network topology, as is easily recognizable by the representation in the figure below, on the left. This contrasts with the illustration on the right, where devices instead connect directly, eliminating the need for a central entity. Notice that all nodes are still somehow reachable. In this case, devices can act as routers and forward traffic for others, enabling the content to hop between them until it reaches a destination. But networks may still be hybrid. If infrastructure is available, mesh networks can take advantage of it and in many cases improve their functionality.



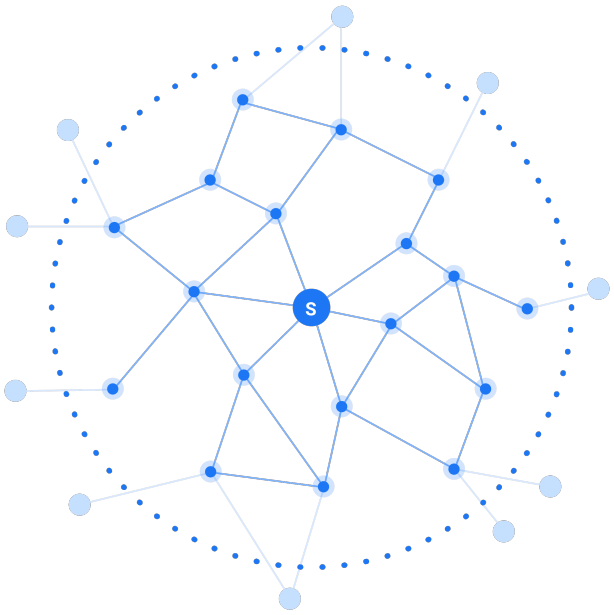
The first obvious advantage of mesh networks is eliminating a problem known as **single point of failure**. Continuing on the previous example, consider what would happen if the access point in the center of the star was compromised. As a central node to the network, the router going offline brings the network down as a whole. Now consider the same scenario in the mesh, and notice that any link breaking does not compromise the network entirely. Rather, this change to the network's fabric makes it more versatile and resilient. This is true in terms of scale manyfold. That is, although only local (LAN) connectivity is being considered, these conclusions can easily be extrapolated to any scale, such as telecom infrastructure deployments and even satellites, for example. In short, any type of network can benefit from this upside. This becomes even more relevant in situations where compromises are expected, such as catastrophe scenarios, where infrastructure is often destroyed.

Mesh networks present themselves in one of three types: proactive, reactive, or hybrid. These classes of protocols differ mostly on the paradigm they use to perform network discovery, which ultimately has consequences on performance and scale. Proactive protocols keep a constant discovery process, having

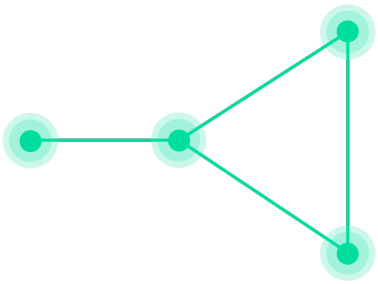
nodes inform each other of topological changes. This type of discovery responds well to link breakages because the network is capable of converging from a failure, but it also incurs overhead. This contrasts with the reactive approach that instead establishes routes on demand and thus does not incur as much overhead. Reactive networks scale better, but take more time to establish connections because paths may not be known a priori. Hybrid protocols can switch between the two approaches on a per-packet basis, and thus adjust better to conditions where either technique is favorable. The Hype SDK uses a hybrid protocol, and is thus capable of both types of network discovery.

In **proactive** routing nodes periodically exchange topological information, notifying each other of what other devices are reachable on the network. This information is exchanged through control packets, a specific type of packet that nodes use to flood the network with topological metadata. This process enables nodes to keep up-to-date routing tables of paths to other reachable nodes, which is why such protocols are often also called table driven. Such information also enables the network to quickly heal from link breakage, by relying on the knowledge of alternative paths to the same destination. Depending on the type of protocol, such tables usually keep next hop information along with other path metadata attributes, such as cost and configuration. For a specific node, the set of participants that it is proactively aware of is called its proactive area, as illustrated in the following figure. The advantage of a proactive approach is that optimal paths to nodes on this set are known, and thus communication links are immediately available when needed.

"For a specific node, the set of participants that it is proactively aware of is called its proactive area."



There are several considerations that make proactive protocols a challenging effort, especially when mobility is involved. One such case is implementing **loop prevention**, which consists of eliminating cyclic redundancy from the network discovery process. The illustration below shows a typical network loop after its formation. Under this scenario, a naïve implementation of a proactive protocol would consider three different paths from the node in middle having the one isolated on the left as destination, although two of those paths would be redundant. It's noticeable that this setup is very common, and as such solid proactive protocols must consider scenarios such as this one.



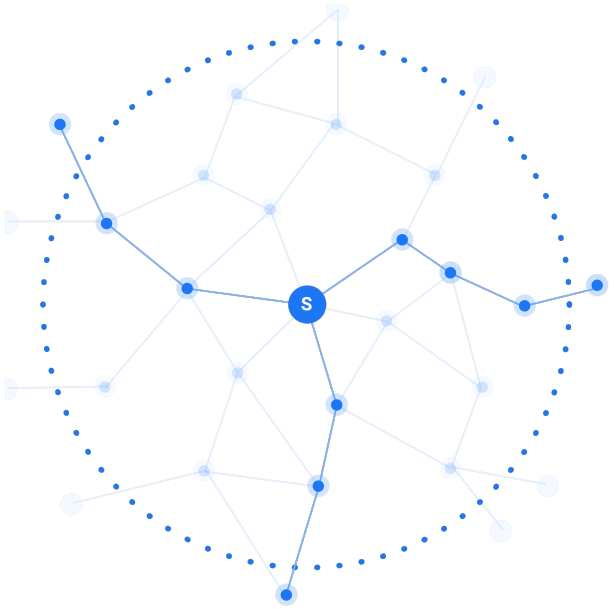
Loop prevention is especially important when links break. For example, consider that the link with the isolated node breaks, due to it moving away, or otherwise going offline. Although pragmatically speaking it's obvious that the isolated node is no longer

reachable, the node in the middle may still consider the other two alternative paths, making it believe that the two can still communicate. The later problem requires network healing, a concept related to making the network's knowledge of its own topology converge when changes occur.

The downside of a proactive approach is that the use of a periodic flooding technique incurs significant overhead. As the number of nodes on the network grows, so does the need for topological maintenance, and thus the amount of control packets circulating. Depending on the protocol, the growth of the number of control packets may get to an exponential factor of the number of nodes, meaning that the network gets congested fast at relatively low thresholds of nodes. For this reason, although proactive protocols are fast at establishing communication links, offer great throughput, and heal with low convergence time, they do not scale very well with the number of nodes. The scale varies greatly with specific deployments and is influenced by factors such as the network's density, node mobility, and other environmental factors. Factors such as these make it hard to precise a number of nodes that can collaborate on a proactive area, although estimations can be given for specific situations.

"Although proactive protocols are fast at establishing communication links, offer great throughput, and heal with low convergence time, they do not scale very well with the number of nodes."

Reactive routing consists of a network discovery and routing technique in which routes are established on demand. The network is maintained only for active routes, thus the topology is not complete and the network conserves resources. In order to establish routes, nodes broadcast route requests, a process that may be further optimized if multicast heuristics are put in place. Route requests are flooded on the network until a path to the destination is found or the request times out. When a path is found, the nodes cooperate to return a route response to the requester and update routing table entries to keep track of the path. At that point the network may proactively maintain the route, depending on the specific implementation. The following illustrates three routes established proactively. The key observation is that other routes are not maintained, which is illustrated by them being grayed out. Because the topology is not complete, reactive protocols incur less overhead, and thus are capable of reaching outside a node's proactive area, giving more range and scale.



With a reactive approach the network is capable of scaling, both in number of nodes and range. The main drawback lies with the time taken to establish routes when first requested. That is, there's an associated latency period between the time the route is requested and the destination is found, which may vary greatly depending on the network's topology, distance, and many other factors. During this period of time, two end nodes may not yet communicate, and must instead sit idly for a route response. This extra delay may be attenuated by the route request packet including the initial message that the originator intents to send. The fact that the packet is flooded on the network, however, creates a tradeoff in which the payload may cause network congestion, a problem that is exacerbated by scaling with the number of route requests and participating nodes. With reactive protocols it makes more sense to use the number of ac-

tive links as a measure of scale, rather than the actual number of participating devices. Although estimations can be given for specific deployments, this fact must not be disregarded.

"With a reactive approach the network is capable of scaling, both in number of nodes and range. The main drawback lies with the time taken to establish routes when first requested."

It's generally considered a rule of thumb that cabled networks are more efficient than their wireless counterparts, but mesh networks can be implemented over any kind of communication channel. What differs is the topology, not the medium. Actually, they can be implemented over several types of transport simultaneously, in which case they are said to be **multi-transport**. Transport agnosticism gives devices the power to interface these transports together, a technique called **bridging** being illustrated below. The device on the back is equipped with Bluetooth Low Energy and the device on the front with Wi-Fi Direct. The device in the middle uses both, enabling the other two to communicate. This paradigm is possible with any set of transports, including, even, the Internet, thus mimicking a similar concept called tethering, in which Internet is made available over radio technologies such as Bluetooth.

Bridging is achieved with two hops, the simplest form of a mesh network, but things do get more complicated. As the number of hops increases, so does the network's complexity, and not necessarily in a linear way. Perhaps surprisingly, the first threshold is quite difficult to overcome. By increasing the depth by a single hop, up to 3, the network starts forming loops. Loops harm the network by adding redundancy. Without proper mechanisms for loop prevention, the content may end up circulating endlessly without ever finding a destination, adding overhead without function. The problem is further aggravated with node mobility. Nodes moving on the network cause connections to drop and the need for the network to recover lost paths or, in other words, for it to heal itself. Mesh protocols that are capable of that are said to be self-healing.

"As the number of hops increases, so does the network's complexity, and not necessarily in a linear way."

At least to some level, all **self-healing** protocols conform to the same mechanisms. The network is monitored, by having the system analyze key indicators that are compared with predetermined thresholds, while observed for variations. If a link between two nodes degrades, how should the system react? After detecting the problem, the cause is first diagnosed. For example, two devices could be going out of range, or otherwise be congested by a temporary peak in traffic. The system reacts depending on the cause. In some situations it can also be preventive. Consider an abnormal number of nodes joining a given network. Given that the system is capable of identifying the irregularity, nodes could collaboratively prevent excessive flooding in detriment of fast convergence, thereby avoiding aggravation. It was IBM who first defined the concept of an autonomic system as one that is capable of coping by itself with the intricacies of its lifecycle. According to that study, self-healing is one of the four major characteristics of such systems, along with the concepts of self-configurability, self-optimization, and self-protection. Collectively, these attributes are referred to as **self-CHOP**.

Taxonomical studies often classify mesh networks in accordance to a wide variety of characteristics, such as size, location, routing protocol, or wirelessness. An important distinction to make, and one that impacts many considerations on the deployment of a mesh network, is whether mobility is involved. For the most part, the autonomic properties of a network gain more relevance when the network's components are not stationary, in which case the network is said to be dynamic. Mobility incurs maintenance overhead, in the sense that links between participating nodes break often, with devices moving close and away from each other frequently. Depending on the mobility model, managing such networks may be challenging. It's notable, however, that even static networks benefit from the same autonomic characteristics. In that case, the network may not need to heal from link breakage due to mobility, but other factors are involved, including hardware malfunctions, downtime, or congestion. These networks can also be classified in terms of the type of devices involved. Common variations include Wireless Sensor Network (WSN), Vehicular Ad Hoc Network (VANET), or Smart Phone Ad Hoc Network (SPAN). Usually, however, more attention is paid to whether the nodes are mobile, making it more common for researchers and enthusiasts to refer to the broader concept of Mobile Ad Hoc Network (MANET). Classifying the network in this way is transversely imperative, as the type of network impacts choices such as the protocol to use, battery policies, scale, and security considerations alike.

All things considered, mesh networks are as useful as they are challenging. There are many factors that must be considered when deploying a network of this kind, such as the type of routing protocol and network discovery, security, healing mechanisms, and many others. Such factors impact network performance and feasibility alike. Historically, mesh networks have been hampered by limited device capacity, but as they pave their way through a new breed of things, their applicability becomes more and more relevant. They are seen used in a wide variety of scenarios, like private networks, network offloading, Internet reachability, social networking, ambient proximity, and many, many more. Although powerful, mesh is not meant to replace existing topologies, but rather complement them. This type of topology has for long been overlooked, but in the words of Brian E. Carpenter on the Architectural Principals of the Internet:

"Principles that seemed inviolable a few years ago are deprecated today. Principles that seem sacred today will be deprecated tomorrow. The principle of constant change is perhaps the only principle of the Internet that should survive indefinitely."

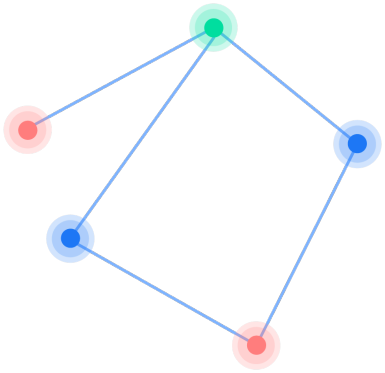
Brian E. Carpenter

State of the techonolgy | Hype currently supports proactive discovery up to 5 hops, which makes it useful for interoperable communications, bridging, social proximity, among others, but not yet for large-scale deployments. The number of hops is being progressively increased as the SDK is optimized in several ways. Reactive routing is not support yet, and is planned for the 3rd quarter of 2018.

Network Segregation

Network segregation is a technique in which devices running different apps can still cooperate on the same network. Consider two devices, each running their own apps. One app uses the FTP protocol for transferring files, while the other uses a general-purpose custom protocol, encoded in a JSON format. Although it's still technically possible for the two apps to exchange data in a raw format, the process would fail as soon as one of them was to try to decode the contents, due to the protocols not matching. Because these two apps cannot "understand" the contents exchanged, they are not capable of communicating. One apparent solution would be to prevent the two apps from corresponding at all, thus creating disjoint networks. Instead, network segregation is a concept that solves this problem without blocking cooperation.

Hype segregates the network using unique app identifiers. The following image illustrates five devices participating on the same network, each painted with one of three colors: red, green, or blue. In this example, the colors represent different apps, and thus different app identifiers. Red devices, for example, are capable of communicating with each other, even though both green and blue devices sit between them. The technique is simple: when an app identifier mismatch occurs, packets do not go up the protocol stack. Using this technique, although devices running different apps cannot communicate directly, they still cooperate in relaying content for each other. With this feature, app developers need not worry about protocol mismatches or unintentionally exchanging data with apps from different vendors.



Network Discovery

Hype uses several techniques to optimize the network. For the most part, this is an automatic process, but for the sake of being abstract, some decisions must be left to the application. It is thus important to understand how devices discover each other, and how to best use discovery features. The first thing to understand is how devices are identified on the network. Hype uses three components for that: an app identifier, a device identifier, and a user identifier. The **app identifier** is used to segregate the network, as discussed in a section of its own. This identifier is always the same. **Device identifiers** are generated and managed by Hype, and used to uniquely identify devices. Although these two identifiers, put together, are already universally unique, Hype also allows setting optional **user identifiers**, which are directly managed by the app. These later identifiers provide the developer with the means to identify a user early in the discovery process. All three are collectively used for discovery and routing purposes.

Lets assume for a second that the app shows a list of discovered devices to the end user; which of the three identifiers, or any combination, should the user see? As all of them are just buffers or numeric values, none seems appropriate. Hype solves this problem in two ways. One, enabling the app to manage user identifiers, giving it the opportunity to book-keep associated user metadata. One option would be to use some sort of local database with user mappings. Two, the concept of announcements, consisting of a small amount of custom data that is exchanged during an handshake. There are two differences between these, specifically their size limitations and the times at which they are exchanged. User identifiers are limited to 31 bits, while announcements have varying, yet more relaxed, restrictions. User identifiers are also discovered earlier, when an instance is found, and are thus always available. Announcements are only exchanged during handshakes, when an instance is resolved.

Properly grasping the difference between the two implies understanding how instances are discovered. On a proactive protocol, devices pass along global identifiers and not much more metadata, keeping the overhead to a minimum. When a new identifier is discovered, Hype triggers a **found notification**, meaning that at this point only the instance's identifiers are known. It's notable that, if they are to engage in communications, the two devices still need to perform an handshake. This is a process that consists of negotiating connection parameters, such as public encryption keys, announcements, and other con-

figurations. In order to save overhead, Hype does not autonomously perform the handshake, meaning that, when instances are found, user identifiers are available, but announcements are not. This is true until the handshake is explicitly requested, a process known as **resolving an instance**. The rationale for this is simple; instead of blindly performance an handshake with every instance it finds, Hype delegates this responsibility to the app, the one in better conditions to decide whether an instance is interesting for communicating. If the app decides to perform the handshake with only the subset of found instances that the user actually wants to communicate with, precious network overhead is saved.

"**In order to save overhead, Hype does not autonomously perform the handshake.**"

We've often seen developers solve this problem by exchanging initial messages, usually containing user names, profile pictures, or some other form of user metadata. With announcements, developers can participate on the negotiation process, which is better than the previous alternative. The reason for this is because Hype amortizes the announcement delivery by caching it in intermediary devices. That is, when two devices are performing handshakes in mesh, routers will forward announcements because they have previously cached them. This means that negotiation information needs not travel the network from origin to destination, as messages do, imposing much less overhead. However, at negotiation time, the devices have not exchanged cryptographic keys yet, making it impossible to encrypt its contents. If the data being announced is sensitive, perhaps the message exchange alternative is the way to go. Nonetheless, keep in mind that there's a trade-off.

State of the techonolgy

It is true that announcements reduce network overhead when compared to the initial message alternative, but if they grow too much they may cause bloat. This makes it a bad idea to allow announcements with significant size to circulate on the network. Current releases support up to 255 bytes of announcement data, a limit that is expected to grow to around 900 bytes during the first quarter of 2018. It's not known yet what the final limitation will be, as the performance is still being profiled and optimized. All other features are already available.

THE HYPE SDK

Multi-transport

Multi-transport is a concept that refers to a device's ability to maintain more than one communication technology simultaneously. This is quite common, especially in high-end devices, which nowadays often support all Bluetooth, Wi-Fi, and Ethernet, among others. Other less common possibilities include WiMAX, ZigBee, and SigFox. The Hype SDK does not implement transports of its own, but rather relies on existing channels and implements mesh technology on top of them, one of the key points enabling it to be so easily deployable. Currently, Hype supports the most common types of transports, namely Wi-Fi Infrastructure, Wi-Fi Direct, Bluetooth Classic, and Bluetooth Low Energy, defined as follows:

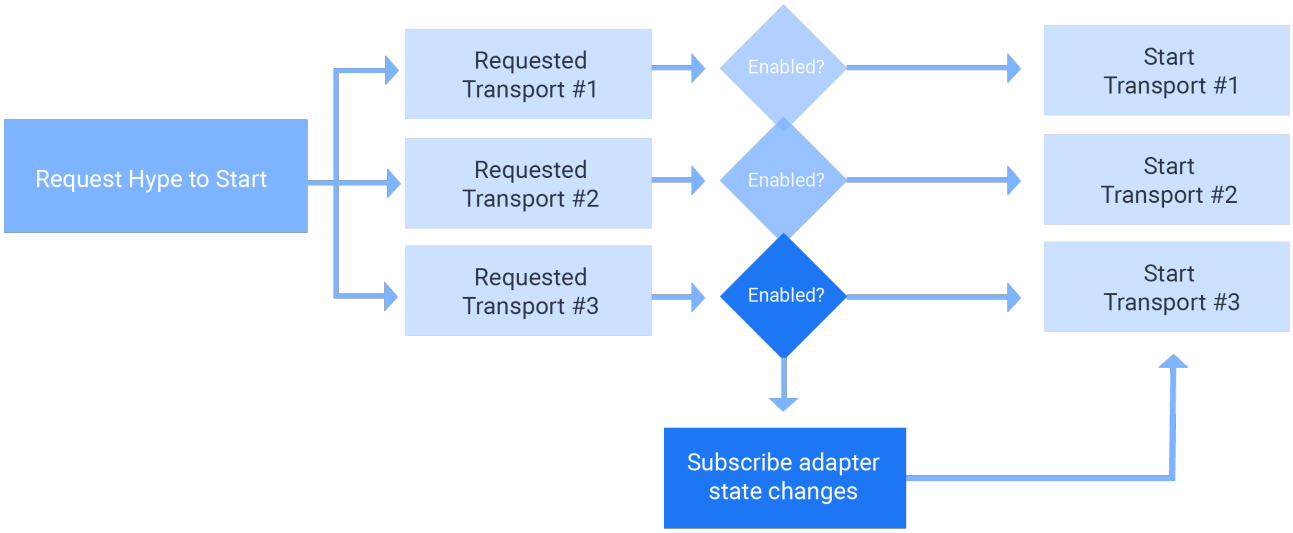
- **Wi-Fi Infrastructure** refers to Wi-Fi (IEEE 802.11a/b/g/n/ac/ad) operating in infrastructure mode. In this mode, devices connect to a central device, called an access point.
- **Wi-Fi Direct** is a Wi-Fi operating mode in which devices connect directly to each other, without the need for additional infrastructure.
- **Bluetooth Classic** is used to refer to Bluetooth BR/EDR (IEEE 802.15.1 and revisions).
- **Bluetooth Low Energy**, or Bluetooth Smart, was first introduced with Bluetooth 4.0, and is meant as a battery-friendly alternative to Bluetooth. Although their names resemble, the two technologies are incompatible and operate in very different manners.

The Hype SDK is capable of managing several transports simultaneously and the developer is given the choice of which transports to enable. In order to make that decision, the developer is faced with a trade-off, meaning that there are both pros and cons to be weighted. On one hand, multiple transports give the device more uptime, better interoperability, and an extended set of alternatives to communicate. On the other, extra transports add to battery consumption, reducing the device's lifetime. What must usually be considered is the type of device, along with the purpose it serves. Enabling several transports is usually not a problem for wall-powered devices, but when considering the realms of mobile and IoT this feature should be used with care.

> "**Multiple transports give the device more uptime, better interoperability, and an extended set of alternatives to communicate.**"

Requesting the SDK to use a specific transport does not necessarily mean that it effectively will. Notably, the operating system and the end user have control over which transports are enabled on the device, a choice that Hype does not attempt to override. However, when a requested transport is, or becomes, disabled, the SDK subscribes to adapter state changes and automatically attempts to start it when it becomes available again by the adapter changing

state. This decision is made transport-wise, meaning that enabled transports are started while disabled ones are not. As transports are toggled, Hype detects those changes and alters its behavior according to the device's state. This transparent management of active transports removes from the developer the need to worry about the device's state and whether specific transports are enabled or not. This process is illustrated by the diagram below.



State of the techonolgy | As Hype evolves, not all transports are equally developed. Consequently, the implementations differ in terms of quality. Furthermore, transport implementations often use system frameworks, meaning that the SDK inherits bugs and quirks from the system. At HypeLabs, we invest a great deal of effort on fixing issues related to transport layers and working around these quirks, to whatever extent is possible. As we work our progress through these, we realize that understanding the state of the technology is important, and that many technology-related decisions depend on factors of stability. For this reason, known issues are being documented as an informational listing of the SDK's open issues with regard to transport implementations. This list is expected to be available during the 1st quarter of 2018, in the hope of shedding light on current limitations.

Internet Reachability

There are many reasons that can cause a device not to have some form of Internet connectivity, such as not being physically enabled to, or otherwise being out of range. By allying the concepts of mesh networking and multi-transport together, Hype can reach the Internet in many ways. Consider the illustration that follows. The device on the left is only equipped with Bluetooth technology, and is thus incapable of accessing the Internet. However, it is connected to a second device, which, besides also having Bluetooth, has a Wi-Fi connection to the Internet. This device may optionally share its connection, meaning that the one on the left is also capable of reaching the Internet. This setup is showing a third device, also using Wi-Fi, and in a geographically different region from the other two; nonetheless, all three are capable of communicating. This concept is called **Internet reachability**. This technology appears in two forms. One of them consists of enabling two Hype-powered devices to communicate in a peer-to-peer fashion, which is the scenario being illustrated. However, a second situation to consider is enabling these devices to directly access the World Wide Web. Considering the same scenario, that would enable the device on the back to access URLs, download files from a remote server, and the likes of it, all through using Bluetooth connectivity and the Hype SDK.

Both forms of connectivity are demanding in their own way. For example, in a common scenario, devices are behind a Network Address Translator (NAT), which masquerades IP addresses on private networks. Because device IP addresses are private, connecting to them implies the use of techniques related to the concept of NAT traversal, such as TURN, STUN, or ICE. It's notable that many such techniques require the assistance of servers outside the network, meaning that, in order to establish a peer-to-peer connection, the devices first need to communicate with a central server. Hype does this automatically, in a way that is transparent to the developer. In some situa-

tions, traversing a NAT takes time; during this period, the two end devices cannot communicate directly. To solve this problem, Hype may temporarily relay traffic on a server, making it possible for the devices to start communicating immediately. This form of communication is not ideal because it's not genuinely peer-to-peer, but it does serve as a temporary replacement while the devices establish a direct connection. Notably, encryption is still end-to-end, meaning that the central server is still not capable of interpreting the contents being exchanged between any two devices.

One final consideration regarding peer-to-peer Internet connectivity is concerned with how the device discovery happens. Depending on the scale of the deployment, a proactive approach - one in which the server would dump the table of accessible devices - may not scale. For this reason, Internet device discovery is reactive. That is, developers using the SDK may query the server for a route to a device, which will respond according to whether the route exists. How this happens is also transparent to the developer. In fact, if several transports are available, Hype may query all or a subset of them, including the one that provides Internet connectivity, when looking for a valid route to a requested device. The rest of the process is described in Multi-transport, as it consists of a standard reactive protocol.

"If several transports are available, Hype may query all or a subset of them, including the one that provides Internet connectivity, when looking for a valid route to a requested device."

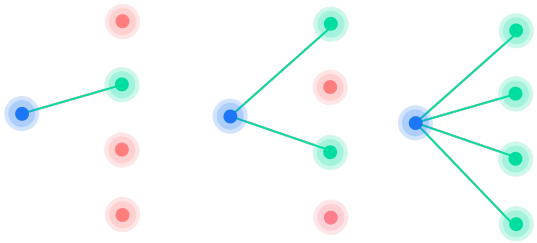
The way that an Internet connecting is being made accessible is of relevance, begging questions such as whether devices should still share connectivity if it was to happen over a mobile data plan, possibly bearing costs to the user. In fact, sharing a data plan without the user's consent is considered illegal in many countries and bears legal consequences, meaning that Hype does not enforce it in any way. Rather, this choice is left to the developer. Hype provides API entry points to set options of whether specific types of Internet connectivity should be shared. The responsibility is thus left to the developer, which may, in turn, delegate to the end user, when applicable. Nonetheless, Hype is aware of how the Internet is being made reachable, and prevents the sharing from happening when developers - and, consequently, end users - do not explicitly indicate that it should.

State of the techonolgy

Internet reachability will be implemented in three phases. The first consists of communicating through a central server, capable of relaying traffic on behalf of the two end devices. This feature is not genuinely peer-to-peer, but it already effectively solves the problem. This feature is planned for the 2nd quarter of 2018. The second step consists of direct peer-to-peer connectivity, already implementing NAT traversal and reactive discovery, in turn planned for the 3rd quarter of the same year. Finally, Hype will allow for Internet reachability by the 4th quarter, allowing devices to access the Internet, even when not communicating directly with other Hype devices.

Communication Methods

Depending on specific use cases, different types of communication methods may be necessary. Sending a message from one device to another, also known as **unicast**, is by far the most common scenario, but there are situations under which this might not be enough. For example, a social app may have the need to create group chats, having messages with several destinations, rather than just one. This concept is referred to as **multicast**. Another example could be that of an app dedicated to rescue operations, in which an SOS could be meant to reach as many devices as possible. In that situation, the message is delivered to everyone, also known as **broadcasting**. The picture that follows illustrates all three concepts.



Hype supports all unicast, multicast, and broadcast. Although the later, multicast and broadcast, can be implemented over unicast, it is considered extremely inefficient, and instead Hype's protocols optimize the delivery in many ways. Actually, the higher the number of destinations the more relevant that fact becomes. It's also notable that the concept of network segregation is never broken. If different apps are cooperating on the same network, even broadcast messages will not be delivered to apps that differ from that of the originator. In fact, what can be seen as broadcasting from the app's point of view might rather be multicast from the SDK's perspective. This is true because Hype only propagates a subset of its knowledge of the network's topology to the app, and hides participants implemented by disparate vendors. In short, if different apps are cooperating on the same network, Hype will take caution not to deliver the content cross-app.

State of the techonolgy | Currently only unicast is supported, but, notably, broadcast and multicast can be achieved by replicating the same message to all destinations. Although this is considered inefficient, it should suffice for small-scale deployments. Both features are planned for the 2nd quarter of 2019.

Network Offloading

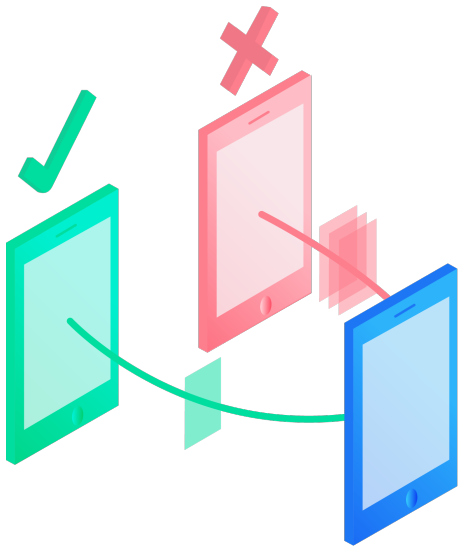
When delivering content over a network, the choice of path bears consequences. Specifically, choosing the best path is a function of multiple variables: overhead, congestion, stability, battery, etc. Hype provides several API entry points for sending data. It might be noticeable that these lack parameters for specifying which transport to use, but there's a reason for that. Choosing the transport when sending data is a common request from developers using Hype, but situations exist under which the feature could be misleading. This happens in mesh. Consider a multi-hop route between two devices. It might seem tempting for a device to use its fastest transport, say, Infrastructure Wi-Fi, to reach another device, even if a Bluetooth alternative exists. The thing is, on a multi-hop scenario the rest of the path is not known, and it may contain bottlenecks, greatly reducing the path's viability. In fact, making a choice of best path using the transport as criteria is not possible without knowledge of the topology ahead. Rather, Hype scores the connections, allowing it to choose the best according to a probabilistic measure of performance.

Routes are scored using real-time profiling metrics, based on the developer's choice of heuristic (i.e. performance vs. battery efficiency). The SDK uses a probabilistic algorithm to decide which transports to use on a per-packet basis. The scores are computed based on a link's response to traffic and a device's knowledge of its surrounding topology; that is, traffic circulating on the network provides an opportunity for profiling the link, even if the traffic is meant for topological maintenance. This means that Hype does not choose a transport based on its type, but rather on key performance indicators. To do so, several factors are taken into considerations, such as battery status, link stability, a function of packet loss, and congestion, among others. One key property to notice is that this algorithm is conservative, and that it acts in the best interest of the network by preferring links that hop on devices with better battery statuses. By worsening such scores, Hype extends the lifetime of the network, by fairly dividing the effort over available participants.

> "**Routes are scored using real-time profiling metrics, based on the developer's choice of heuristic.**"

Choosing links based on their performance ultimately causes **load distribution**. As a link becomes congested and its score worsens, it turns into a less probable candidate of choice, and thus less traffic is sent on its way. It so happens, as other routes are cho-

sen in its place, the scores tend to even out, creating a more ideal distribution of network load. As the traffic sent over one path overloads it, Hype diverts part of it, creating balance between the two. This also applies to local infrastructure. If several access points are available and a subset of those is connecting a large portion of the devices, the SDK may divert the traffic to different access points, outside of the overloaded subset. In practical terms, this means that Hype helps reducing the costs of infrastructure deployment, by relieving the existing infrastructure, pushing it to its full potential. It should come as no surprise that the costlier the infrastructure, the greater the benefits.



State of the techonolgy

Notably, network offloading cannot be considered a feature per se. Rather, several of Hype's features contribute to that effect. Hype already uses technologies that do not rely on the infrastructure and uses basic link scoring, two mechanisms that help offloading the network. Load distribution, however, can only be achieved with a fully fledged scoring mechanism, currently planned to be complete by the 2nd quarter of 2019. Until then, the system will be progressively enhanced and the subject of frequent updates.

Battery Efficiency

Maximizing throughput and optimizing for battery savings are two mutually exclusive concepts. Technologies that provide great throughput (e.g. Wi-Fi) also drain battery faster and, conversely, low throughput transports do the opposite (e.g. Bluetooth Low Energy). Hype introduces the concept of **policies**, enabling developers to choose what Hype should optimize for. Devices that need to maximize throughput will have higher battery consumption, while optimizing for battery savings has the contrary effect. The SDK implements several types of battery optimizations, by minimizing battery-heavy operations. One such operation is scanning the network for other devices, an operation that, when withheld, provokes an increment in network discovery time, but ultimately extends the device's lifetime. Also, the SDK implements a scoring system. Each route is ranked according to cost, and costs are calculated according to several factors, including power source and power supply. As such, devices with low battery capabilities will have higher costs of traversal, forcing Hype to prefer relaying data while hoping on more battery-capable devices. The choice of transport is also important. In extreme cases, the developer may select only low-power transports, causing Hype to not use other types of transport at all.

"**Maximizing throughput and optimizing for battery savings are two mutually exclusive concepts.**"

State of the techonolgy | Currently Hype does not perform any kind of battery management. Instead, all services are enabled all the time, which constitutes considerable drainage. Consequently, policies have not been implemented yet either. Battery drainage optimizations are being gradually implemented into the SDK, and policies are planned for 1st quarter of 2019.

Progress Tracking

When sending content, it is commonly desirable to track how much of the data has been delivered and when. Devices are easily capable of identifying when the data has been written to the network (i.e. left the device), but if the data is hoping on other devices before reaching the destination the process is not as linear. The later implies acknowledgements back from the destination, which may need to traverse the path in reverse - or even a different one - back to the originator. Acknowledgements are performed on a packet basis, meaning that larger messages must be tracked in parts. As the content is being sent, end users often expect a progress bar indicating how much of the operation has been completed. For the sake of usability, Hype provides the option to enable progress tracking when sending data. When enabled, the SDK notifies the application when the content has been delivered. That being said, this feature causes the destination to send acknowledgements back, an operation that incurs extra overhead, so it becomes obvious that the decision to enable it should be a product of need. When this option is not enabled it's not known by the originator whether the content was delivered at all; however, in situations under which that is not necessary, developers should prefer to disable the option and avoid the increased overhead.

"**Acknowledgements are performed on a packet basis, meaning that larger messages must be tracked in parts.**"

THE HYPE SDK

Security

Usual network security paradigms involve some form of centralism. For example, authentication is a process that commonly requires querying a database, or some sort of permission registry. Because mesh networks are decentralized in nature, it becomes obvious that a paradigmatic equivalent approach is not possible. To make matters worse, common deployments involve low-end devices, with highly constrained resources and often limited in terms of bandwidth, memory, and processing power. When working over wireless mediums, the problem is further inflamed by an attacker's ability to sniff traffic from the air. Unless cryptographic mechanisms are put in place, the contents being exchanged between devices are plainly visible to their surroundings. As in a mesh network devices forward content for each other, it must be asserted that those assuming the role of routers are reliable so not to observe or tamper with the data. Was that not to be true, and the network would be highly sensible to man-in-the-middle attacks, a specific type of hacking in which an attacker mediates a conversation.

The first step in securing communications is to establish encrypted end-to-end links between two devices. Hype uses the Diffie-Hellman, allowing any two peers to safely exchange data without the worry that attackers may read or tamper with the contents. This is true even in mesh. This method is used to encrypt the data in a way that only the two end devices can interpret. Plus, the contents are validated at the destination, meaning that if attackers attempt to tamper with the data, the receiver will be aware of it and able to conscientiously protect itself.

"Hype uses the Diffie-Hellman, allowing any two peers to safely exchange data without the worry that attackers may read or tamper with the contents."

Encryption may be the first thing that comes to mind when discussing security, and albeit an important part of it, it is not the only requirement for secure networking, by far. Encryption already covers confidentiality, non-repudiation, or integrity, but not the likes of authentication, which is aggravated by the fact that allowing untrustworthy devices to join the network opens all sorts of security holes. The key characteristic of mesh networks is having a flat hierarchy, meaning that no central entity is regulating access to the network. Hype solves this problem using digital certificates and a means of challenging other devices for authenticity. That is, one device issues an authentication challenge, to which another must respond with the expected scheme. The two devices exchange certificates, which have predetermined expiration dates. The communication link will only be maintained if both certificates are valid and the challenge is met, or otherwise failing device(s) will not be allowed to join the network.

It's notable that the SDK will refuse to run without a valid certificate, meaning that one must be installed on the device at all times. There are two ways that this can happen. One, the device accesses the Internet once, authenticates with a server, and, in case of success, a certificate is downloaded and securely stored on the device. Two, the certificate is manually installed on the device when the app is first deployed. The former imposes a requirement for the device to have periodic Internet access, every time a certificate expires or is nonexistent. This is ideal when the app is distributed to the public, such as through means of online stores. The later is only an option when the developer has control over the hardware, as it requires direct access when installing the app binary. This is valid when setting up private networks, or in situations under which the hardware is fully controlled.

There are three important periods to understand regarding certification. Initially, the user provides authentication details that are validated with a central server, or the certificate is installed directly on the device. The certificate has a predetermined validation period, during which it can be used to join the Hype network. When the certificate expires, a new one is required, implying a renovation process. There are cases where the device does not have Internet access at the time that the certificate expires, so Hype may download the next certificate in advance. This initiates a grace period, during which the certificate is installed but not used. As such, when one certificate expires, the next one is already installed. As a direct consequence, there's no such period during which the device is left out of the network. This process is repeated every time a certificate is nearing its expiration date, given that "nearing" stands for whatever period is configured on the HypeLabs dashboard. Developers can configure the validity and grace periods of certificates issued on their app's behalf.

"As a direct consequence, there's no such period during which the device is left out of the network."

State of the techonolgy | Certification and encryption are both planned for the 1st quarter of 2018. Authentication too, but that topic is described in a section of its own. Manually installing certificates is not placed on the roadmap yet, as the plan for IoT is still being designed.

Authentication

Hype provides the necessary mechanisms to ensure that unauthorized users do not gain access to the network. For that effect, there are two app modes to consider, development and production. These two modes serve different purposes. The former, development, is used as a quick way to get started, but is ultimately considered insecure. The later, production, uses OAuth 2.0 to certify devices and is the method of choice for app deployments. Both modes refer to user authentication, but the mechanisms differ. What they have in common, however, is the use of an authorization flow. Authorization is the process by which devices prove they are authorized to acquire digital certificates. Every time Hype is requested to start without a valid certificate being installed on the device, either due to non-existence or expiry, the SDK will prompt the app for an access token, by means of a call to a state observer method designed for that effect. This is where the two app modes begin to differ.

> "**Authorization is the process by which devices prove they are authorized to acquire digital certificates.**"

When in **development** mode, the app statically returns an access token specifically generated for it by the HypeLabs dashboard. This token can be found under the app's settings. This mechanism only certifies up to ten devices, and only for periods of 24h, after which they expire. These may seem like harsh requirements, but this is justified by the fact that this method is not considered secure and is thus not suitable for deployment. Instead, its purpose is to allow testing Hype without the effort of creating an OAuth 2.0 authorization server. The SDK uploads this token to the HypeLabs' dashboard, which, by checking that the app is in development mode, asserts whether the tokens match. If they do, and the maximum number of devices has not been reached, the certificate is emitted and installed on the device. It's noticeable that development certificates are not compatible with production ones, and test-certified devices are still not capable of joining the network. Think of these as two disjoint networks, one that scales with numerous amounts of devices, and a test one that doesn't.

Deploying a **production** version of the app is somewhat more complex. The static token used during development is no longer available, as toggling the authorization method disables it. Instead, the tokens must be provided by a third party. The authorization is then performed in three steps, as illustrated in the picture that follows. In step 1, the app forwards the SDK's token request to a third party authorization server. Step 2 consists of the SDK posting the retieved access token to the HypeLabs server, which, in step 3, validates the access token against a third party server. If the third party validates the access token, then HypeLabs will issue a certificate for it, using whatever settings are configured for that app, such as expiration and grace periods.

The protocol used in step 1, between the app and the third party, is open for customization at one's discretion. In short, the app should post the user's identifier, plus whatever other credentials are needed, with the purpose of authentication. At this point, the third party generates and stores a one-time access token and returns it to the app, which in turn forwards it to the Hype SDK. Step 2 is automatically managed by the SDK and requires no intervention. During this step, the SDK uploads the app's and the user's identifiers to the server, along with the corresponding access token retrieved in the previous step. At this point, the SDK is merely waiting for the validation process to complete, before proceeding with responding to a pending start request. Finally, the HypeLabs server checks the app's settings as configured on the dashboard and forwards the request according to them. During step 3, the third party returns a success HTTP status code in case of success, or otherwise the authorization credentials will be rejected and the device will not be certified, resulting in Hype failing to start.

State of the techonolgy | Authentication and authorization are both planned for the 1st quarter of 2018. Part of the work is on the developer's side, due to the need to implement an OAuth 2.0 authorization server. However, demos, samples, and templates are being developed to minimize the impact of that effort.

03

Use Cases

USE CASES

Internet of Things

Internet of Things

Affordably connecting things

A mind-blowing diversity of IoT devices makes connecting them a hard and expensive task. Hype reduces the complexity and costs of IoT connectivity by creating local networks of devices with cloud access.

Smart Cities

Connecting things to people

Technology-driven cities are more efficient and offer better quality of life to citizens. Hype provides core connectivity for smart automation on all kinds of disconnected devices.

Edge computing

Clustering things

Cheap devices may not offer as much computational power, but clustering several together promises to overcome shortcomings. Hype is the connectivity layer that enables managing devices in groups.

Smart Home and Office

Smart homes need smart connectivity

Smart homes need smart devices, and smart devices need smart connectivity. Hype can connect all sorts of things, and enable the next generation of smart housing and office.

Utility sensor connectivity

Connecting things to people

Technology-driven cities are more efficient and offer better quality of life to its citizens. Hype provides core connectivity for smart automation on all kinds of disconnected devices.

USE CASES

Social Networking

Social networking

Empowering social proximity

Proximal connectivity enables all new kinds of social experiences. With Hype, users can connect, communicate, share, and play, even when an Internet connection is not available.

Dating

Proximity dating on steroids

Why the need for Internet or GPS when people are right there? Hype uses true proximity technologies to enable the next generation of not-so-online dating.

Messaging

The ultimate signal boost

All smartphones are equipped with messaging capabilities, but are rendered useless when a network signal is not available. Hype can still deliver content when the signal goes down, so users need not look for higher grounds in search of a signal boost.

USE CASES

Disaster Recovery

Offline communications

Communication when everything else fails

Mesh networks are more resilient because they don't depend on existing infrastructure and can function even if it's destroyed. Hype is ideal for catastrophe scenarios and rescue operations.

Emergency broadcasting

Connecting to those who can help

Getting help during an emergency is a role of the dice. Hype enables victims to broadcast distress messages to everyone in the vicinity, improving the odds.

Offline Internet Capabilities

Internet when it's most needed

Connectivity infrastructure aches from single point of failure, rendering it useless during catastrophes. Hype content can hop from device to device until it reaches the Internet, even if the local infrastructure is destroyed.

USE CASES

Sharing

Media sharing

Sharing anything, anytime, anywhere

Sharing files, pictures, videos, and music, has never been easier. With Hype's cross-platform abilities, it's now possible to seamlessly share any type of media between devices created by different manufacturers.

Distributed storage

Connecting clusters for distributed storage

Store files on a local cluster instead of a single device, and improve resilience and accessibility. With Hype, participating devices amortize the storage requirements, making it cheaper and safer to create a Distributed Storage System.

Content Distribution Network

Content for disconnected devices

Disconnected devices cannot download content, such as media, or system updates. CDNs are an easy approach to distribute content on a local network, one powered by Hype!

USE CASES

Connecting Events

Conferences, festivals, and sports events

Better connectivity with less infrastructure

Connectivity is usually bad in events with high counts of people, such as music festivals, conferences, and sports events. Hype offloads the infrastructure by redirecting traffic to less congested access points and keeping part of that traffic off the grid, saving money on infrastructure.

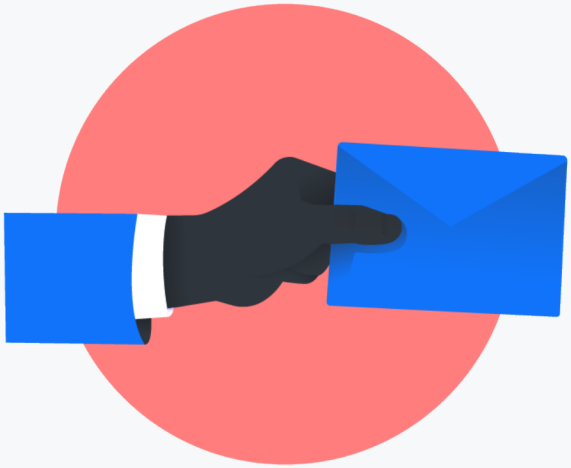
USE CASES

Gamming

Multiplayer offline interoperable gaming

Better gaming experiences with local connectivity

Enable multiplayer proximity gaming without worrying about connectivity. Hype lets you join players with different devices on the same network. It's all sorts of fun!



Contact our sales team

If there’s anything we can help you with, just let us know. We’d love to talk about how we could work together. Find us at:

contact@hypelabs.io

04

Bibliography

[1]Dr. Lawrence G. Roberts (IEEE). *The Evolution of Packet Switching*. URL: http://www.packet.cc/files/ev-packet-sw.html (visited on 01/08/2018).

[2]Felix Richter (statista). *The Not So World Wide Web*. URL: https://www.statista.com/chart/3512/internet-adoption-in-2015/ (visited on 01/08/2018).

[3]Davey Alba (Wired). *Mary Meeker: The Internet's Growth Is Actually Slowing Down*. URL: https://www.wired.com/2015/05/20-years-mary-meeker-says-internet-growth-slowing/ (visited on 01/08/2018).

[4]X. *Project Loon*. URL: https://x.company/loon/ (visited on 01/08/2018).

[5]facebook. *internet.org*. URL: https://info.internet.org/en/ (visited on 01/08/2018).

[6]guifi.net. *guifi.net*. URL: https://guifi.net/ (visited on 01/08/2018).

[7]ninux.org. *ninux.org*. URL: http://wiki.ninux.org/ (visited on 01/08/2018).

[8]PeoplesOpen.net. *PeoplesOpen.net*. URL: https://peoplesopen.net/ (visited on 01/08/2018).

[9](statista). *Annual spending on cloud IT infrastructure worldwide from 2013 to 2017 (in billion U.S. dollars)*. URL: https://www.statista.com/statistics/503686/worldwide-cloud-it-infrastructure-market-spending/ (visited on 01/08/2018).

[10]Benjamin Haas (The Guardian). *China moves to block internet VPNs from 2018*. URL: https://www.theguardian.com/world/2017/jul/11/china-moves-to-block-internet-vpns-from-2018 (visited on 01/08/2018).

[11]Harrison Jacobs (Business Insider). *Here's what Internet is like in Cuba*. URL: http://www.businessinsider.com/is-there-internet-in-cuba-2017-1/#paid-wi-fi-hotspots-are-scattered-through-major-cities-they-are-instantly-recognizable-by-the-crowds-of-young-cubans-gathered-with-their-

eyes-glued-to-an-assortment-of-smartphones-laptops-and-tablets-1 (visited on 01/08/2018).

[12]Dave Evans (Cisco). *The Internet of Things: How the Next Evolution of the Internet Is Changing Everything*. URL: https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf (visited on 01/08/2018).

[13]George Deeb (Forbes). *The Internet Of Things Is Coming, Hang On To Your Hats!* URL: https://www.forbes.com/sites/georgedeeb/2016/04/10/the-internet-of-things-is-coming-hang-on-to-your-hats/#6219559c5240 (visited on 01/08/2018).

[14]Jacob Morgan (Forbes). *A Simple Explanation Of 'The Internet Of Things'*. URL: https://www.forbes.com/sites/jacobmorgan/2014/05/13/simple-explanation-internet-things-that-anyone-can-understand/#79ab0e8e1d09 (visited on 01/08/2018).

[15](Gartner). *Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015*. URL: https://www.gartner.com/newsroom/id/3165317 (visited on 01/08/2018).

[16]Bill Kleyman (InformationWeek). *IoT: How Ready Is Your Infrastructure?* URL: https://www.networkcomputing.com/cloud-infrastructure/iot-how-ready-your-infrastructure/763692586 (visited on 01/08/2018).

[17]Trefis Team (Forbes). *T-Mobile's Big Spectrum Haul Comes With Some Risks*. URL: https://www.forbes.com/sites/greatspeculations/2017/04/17/t-mobiles-big-spectrum-haul-comes-with-some-risks/#1b174fce2587 (visited on 01/08/2018).

[18]Amy Nordrum (IEEE Spectrum). *Verizon and AT&T Prepare to Bring 5G to Market*. URL: http://spectrum.ieee.org/telecom/wireless/verizon-and-att-prepare-to-bring-5g-to-market (visited on 01/08/2018).

[19]Cliodhna Russell (thejournal.ie). *CEO of Web Summit says if dodgy WiFi not fixed 'we won't be in this country much longer'*. URL: http://www.thejournal.ie/web-summit-wifi-problems-1766393-Nov2014/ (visited on 01/08/2018).

[20]Lauren Hurley (BBC). *Tube tunnel wi-fi plan to be launched*. URL: http://www.bbc.com/news/uk-england-london-40107174 (visited on 01/08/2018).

[21]Jerome Henry (Fast Lane). *802.11s Mesh Networking*. URL: https://www.cwnp.com/uploads/802-11s_mesh_networking_v1-0.pdf (visited on 01/08/2018).

[22]Bradley Mitchell (lifewire). *The Range of a Typical WiFi Network*. URL: https://www.lifewire.com/range-of-typical-wifi-network-816564 (visited on 01/08/2018).

[23](3GPP). *The Range of a Typical WiFi Network*. URL: http://www.3gpp.org/ (visited on 01/08/2018).

[24](3GPP). *LTE-M & 2 Other 3GPP IoT Technologies To Get Familiar With*. URL: https://www.link-labs.com/blog/lte-iot-technologies (visited on 01/08/2018).
[25](MQTT). *MQTT*. URL: http://mqtt.org/ (visited on 01/08/2018).
[26](6lowpan). *6lowpan Status Pages*. URL: https://tools.ietf.org/wg/6lowpan/ (visited on 01/08/2018).
[27](The Things Network). *LoRaWAN*. URL: https://www.thethingsnetwork.org/wiki/LoRaWAN/Home#lorawan (visited on 01/08/2018).
[28](IoTivity). *IoTivity*. URL: https://www.iotivity.org/ (visited on 01/08/2018).
[29](Open-Mesh). *Open-Mesh*. URL: https://www.open-mesh.org/projects/open-mesh/wiki (visited on 01/08/2018).
[30](Open Garden). *Open Garden*. URL: https://www.opengarden.com/ (visited on 01/08/2018).
[31](p2pkit). *p2pkit*. URL: http://p2pkit.io/ (visited on 01/08/2018).
[32](Uepaa). *bridgefy*. URL: https://bridgefy.me/ (visited on 01/08/2018).
[33]Malek M." "Milic B. "Properties of Wireless Multihop Networks in Theory and Practice". In: *Guide to Wireless Ad Hoc Networks*. Ed. by Woungang I." "Misra S. 2009.
[34]Edward C. Whitman (Undersea Warfare). *SOSUS: The "Secret Weapon" Of Undersea Surveillance*. URL: https://www.public.navy.mil/subfor/underseawarfaremagazine/Issues/Archives/issue_25/sosus.htm (visited on 01/08/2018).
[35](DARPA). *DARPA*. URL: https://www.darpa.mil/ (visited on 01/08/2018).
[36]Taieb Znati" "Kazem Sohraby Daniel Minoli. *Wireless Sensor Networks: Technology, Protocols, and Applications*. 2007.
[37]Lincoln Laboratory. *Distributed Sensor Networks*. 2007.
[38]*The Evolution of Wireless Sensor Networks*. 2013.
[39]Tibi Puiu (ZME Science). *Your smartphone is millions of times more powerful than all of NASA's combined computing in 1969*. URL: https://www.zmescience.com/research/technology/smartphone-power-compared-to-apollo-432/ (visited on 01/08/2018).