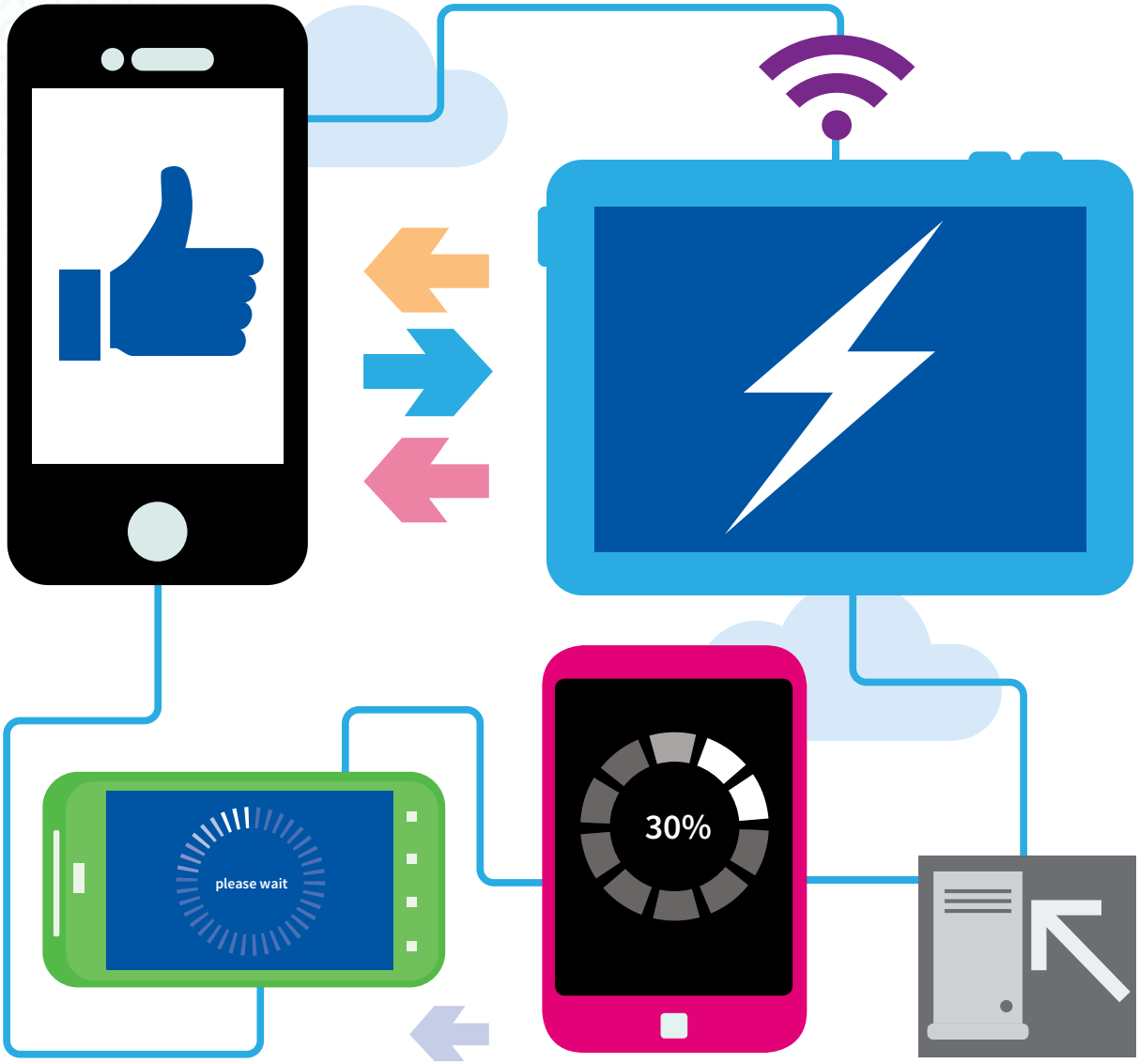
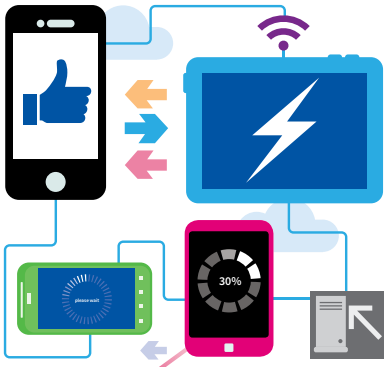
PUSH TECHNOLOGY



Reactive Apps:

What are they and why care?

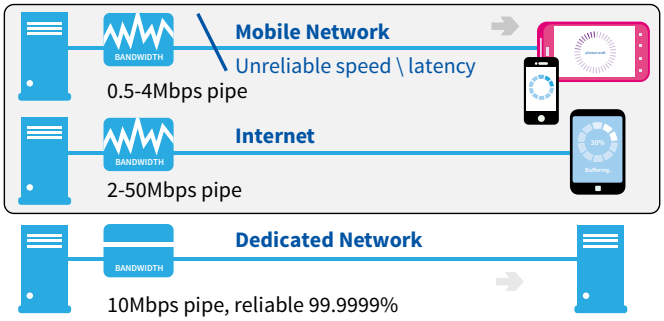


Contents

Introduction

We all know the nightmare app – it's the one that works, sometimes. Instead of offering a great experience, it frustrates users. Even if the nightmare app works, sometimes you wish you hadn't bothered because the quality of service is poor. It struggles with connectivity, reacting badly to lower levels of connectivity from 3G to Edge to GPRS to nothing at all and then back again. It cannot cope with demand during peak periods.

The problem is this is where app users are lost. Research shows that users will not tolerate problematic mobile apps, 79 percent would retry a mobile app only once or twice if it failed to work the first time.



It isn't necessarily the app at fault (although there are always options for optimization). Most often, the network is the problem. Many apps are developed on uncontended networks, where the connection is completely stable and reliable. In the real world, apps are deployed over networks of variable quality such as the Internet or mobile networks. And a mobile network is a very different place to a dedicated network – speed is unreliable, latency is extremely variable and connectivity may or may not be there.

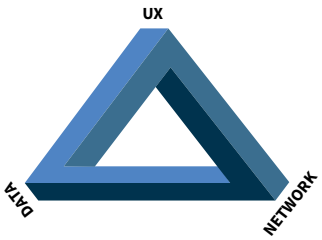
A mobile network is a very different place to a dedicated network – speed is unreliable, latency is extremely variable and connectivity may or may not be there.

Users want interfaces that are engaging. What makes an engaging user interface? Immediacy, interactivity, responsiveness to what users are actually doing in the app. Users want applications that are reactive.

This paper will discuss our term, the 'Righteous Reactive Triangle', define what a reactive app is and how to use the reactive model to optimize the network, data and the user experience.

The Righteous Reactive Triangle[1]

'The Righteous Reactive Triangle' is that engaging apps are a combination of the user interface, the data sent and the network that data is sent over.



It's all about the Network

Engaging apps are all about the network. We are not talking about an app like Angry Birds because it is a local application, it makes no use of the network. Instead, we are focused on real-time interactive apps such as sports betting or private trading where the network is used extensively.

Engaging apps interact with something at the backend. The quality of that interaction depends on connectivity and latency including:

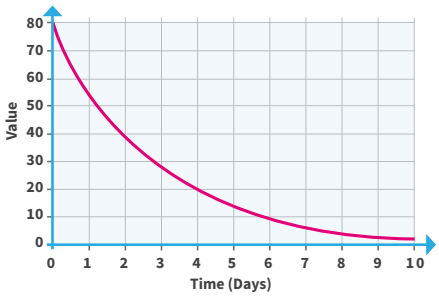
- That data gets through at all
- The time it takes that data to get through
- The time to process that data
- That the data returns
- The time it takes for the data to return

[1]*coined by Push Technology's Dr. Andy Piper*

The problem is that the network is what it is – a precious resource that cannot be controlled. You cannot really do anything about the network.

It's all about the Data

Data is another story. You can control how much data is used and when it is used. This depends on how much is transmitted and how frequently it is transmitted. The good news is that you only need to deal with what users can see. Seeing is what matters for interactivity. An app shouldn't push or pull data that is irrelevant to an end user. For example, in sports betting, if a user is only interested in the oddson four specific horse races, only request the data for those races, not every race. Sending every race wastes both network bandwidth – in sending unnecessary data – and client CPU – through the discarding of that data.



Data also has a time dimension. Highly interactive apps require timeliness of data and the more recent the data, the more valuable that data. The longer the time, the lower the value of that data decays.

There are obviously different kinds of data that may not be time sensitive, but with very interactive apps like sports betting, second screen or gaming apps, delays are very relevant to how valuable the data. It is therefore not just about what is seen, but when it is seen.

But with very interactive apps like sports betting, second screen or gaming apps, delays are very relevant to how valuable the data. It is therefore not just about what is seen, but when it is seen.

It's all about the User Experience

Finally, the 'The Righteous Reactive Triangle' is all about the user experience. For applications to be engaging, they need to be event-driven, scalable, resilient and responsive. This implies an asynchronous user interaction pattern. This is based on the Reactive Manifesto which says:

"…we want systems that are responsive, resilient, elastic and message driven. We call these Reactive Systems.

Systems built as Reactive Systems are more flexible, loosely-coupled and scalable. This makes them easier to develop and amenable to change. They are significantly more tolerant of failure and when failure does occur they meet it with elegance rather than disaster. Reactive Systems are highly responsive, giving users effective interactive feedback."

For more information on the Reactive Manifesto see the appendix.

Applications need to asynchronously send data and asynchronously receive and correlate those responses, rather than use a request-response type interaction where the application waits for a response before sending the next request. The nice thing about this is that, although latency over mobile networks (and the Internet) is inevitable, using an asynchronous approach hides latency and is generally a better model overall (although can be harder to implement).

Using the Reactive Model for a Typical Application

Traditional mobile apps rely upon request-response semantics to fetch data from back-end systems. The app periodically polls the back-end system for data and presents that data to the user. This approach is both heavy on the network and the server CPU, commonly resulting in blocked operations that impact the responsiveness of the overall app. The result? App users experience the infamous spinning wheel.

In a reactive approach, communication with back-end systems is purely asynchronous with no blocking operations performed. The app subscribes to data which is then streamed to the app as it changes. The result? Data is received and processed seamlessly, the user has a great app experience.

Optimize What?

If an app is all about the network, data and user experience, how can you optimize it?

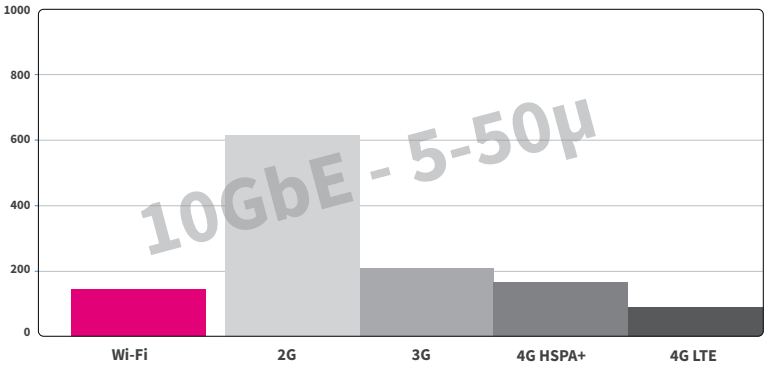
Optimize the Network?

The first question is whether or not the network can be optimized?

The actual, experienced speed of mobile networks varies greatly based upon the technology used as well as the number of network users. The table below shows the theoretical maximum data rates achieved with the current mobile technologies deployed by the telcos. As a mobile user moves, their device can often step between these technologies decreasing and increasing the data rates; steps between 4G, 3G, Edge and GPRS are common.

Technology	Download rate	Upload rate
LTE (4×4 MIMO)	40.750 MB/s	10.750 MB/s
EV-DO rev. C	35 MB/s	9 MB/s
UMB (4×4 MIMO)	35 MB/s	8.5 MB/s
LTE (2×2 MIMO)	21.625 MB/s	7.25 MB/s
UMB (2×2 MIMO)	17.5 MB/s	4.250 MB/s
15×EV-DO rev. B	9.2 MB/s	3.375 MB/s
HSPA+ (2×2 MIMO)	5.25 MB/s	1.437 MB/s
4×EV-DO Enhancements (2×2 MIMO)	4.3 MB/s	1.55 MB/s
HSPA (3.5G)	1,706 kB/s	720 kB/s
1×EV-DO rev. B	1,837 kB/s	675 kB/s
1×EV-DO rev. A	397 kB/s	230 kB/s
1×EV-DO rev. 0	307.2 kB/s	19 kB/s
EDGE Evolution (type 2 MS)	237 kB/s	118 kB/s
EDGE Evolution (type 1 MS)	148 kB/s	59 kB/s
EDGE (type 2 MS)	59.2 kB/s	59.2 kB/s
UMTS 3G	48 kB/s	48 kB/s
EDGE (2.75G) (type 1 MS)	29.6 kB/s	29.6 kB/s
CDMA2000 1×RTT	18 kB/s	18 kB/s
WiDEN	12.5 kB/s	12.5 kB/s
GPRS (2.5G)	7.2 kB/s	3.6 kB/s
HSCSD	5.4 kB/s	1.8 kB/s
GSM CSD (2G)	1.8 kB/s	1.8 kB/s

LTE latency in ms vs latencies experienced on other connection types (smaller is better)



Now, look at latency. It is worse than data rates. Here is a graph of typical latencies for Wi-Fi and various mobile networks. Wi-Fi has a tenth of a second latency delay, which compares favorably with 4G. However, 2G is a different story – a whole half of a second latency which is colossal. If comparing this with normal wired networks, they have latencies in the microsecond range. There is a factor of a thousand difference in latency between regular networks and mobile networks. This makes a tremendous difference to the way applications behave and the way that applications should be written.

So how do you optimize the network? The problem is, you can't. The network is the network. You have to just deal with it. Bandwidth is always limited and costly because of how much you use. Latency is often poor as we illustrated.

So how do you optimize the network? The problem is, you can't. The network is the network. You have to just deal with it. Bandwidth is always limited and costly because of how much you use. Latency is often poor, as we illustrated.

When the Network Lets You Down

The network will let you down – there will be insufficient and inconsistent bandwidth, high latency and loss of network on a regular basis. The network cannot be optimized, but you can respond to it.

This is where reactive apps come into play. Apps must respond to changing network conditions. We call this "mobile sympathy" where you need to be cognizant of what's going on in the network and take action based on those network conditions. Mobile sympathy is the ability to seamlessly handle variations in network capability and still deliver a rich, event-driven user experience.

This is all for the purpose of making decisions, quickly, based on feedback. An example is if you have a server process sending data to your mobile clients. You can actually have a view of how fast that data is getting through and you can take action on the server side to throttle or send less data. This is all about instrumentation. Various products solve these problems. Look for a solution that is focused on what is happening on the network and can take action. Diffusion by Push Technology can help here.

Optimize the Data?

You can optimize data by looking first at data flows and secondly, by using less data – small is beautiful.

Data Flows – The Way You're Sending It

If you're dealing with a server at the back end that is interacting with a front end, compact binary protocols make an enormous difference. You'd be surprised at the number of people that do not put this into practice. Instead, they opt for the worst possible choice – XML or HTTP. There are a number of mobile apps that use this type of interaction pattern; however the problem is the app just will not work over very low-speed networks.

Prefer compact binary protocols

- HTTP not great (300+ bytes/message), XML awful
- STOMP – 100+ bytes/message
- MQTT – 10+ bytes/message
- COAP – 4+ bytes/message

Binary protocols also happen to be faster – scale better

- Serialization is the killer – both for performance and the battery
- Protobuf, SBE, DPT

The good news is there are a number of protocols out there that are binary-based and far more compact for example: STOMP, COAP and MQTT. These protocols are used for Internet of Things (IoT) applications where data rates are hugely important.

Another benefit of binary protocols is they scale better because it is cheaper to encode/decode a binary encoding than it is to encode/decode a text encoding, particularly XML. Encoding and decoding of XML is hugely expensive. This is where serialization is a killer, not only in terms of performance, but with mobile apps, high CPU burden for serialization reduces battery life. To overcome this, you can use tools like Google Protocol Buffers or Simple Binary Encoding which is a new open source initiative. We have our own high-speed protocol DPT that does something similar. The essence is that small binary packets make an enormous difference.

To receive the benefits of compact protocols, you do not want them to be chatty, for example an app that repeatedly uses a request-response model to retrieve data. The set up costs and the overhead of sending messages can be quite high as well as being severely influenced by latency. Instead, you want to make sure that you're sending the minimum possible amount of data in the least amount of time.

Small is Beautiful – Make Information Denser

It is not just the way you send data, but how you optimize its use that can also contribute to app performance.

It is not just the way you send data, but how you optimize its use that can also contribute to app performance. In particular:

- Don't send the same data more than once. Send only differences in data by first sending a snapshot of data followed by changes. It seems an obvious thing to say, but some applications send the same information again and again and again.
- Send less data.
  - Just because you have data doesn't mean you should send it
  - Just because you have bandwidth doesn't mean you should use it
  - Throttle based on the ability for the device and humans to consume
  - Throttling reduces data costs
  - Throttling conserves bandwidth
  - Being reactive means only sending data when you need to, making an app much more efficient

Optimizing the Experience?

Finally, can we optimize the user experience? Yes and the good news is there are many ways you can do this. We are advocates of real-time user experiences. An app is significantly enhanced by making the experience contextual. New APIs from Facebook and Foursquare do not just focus on the data itself, but just as important who is sending the data, where they are and when they send it.

Contextual data often originates from the device itself (location, movement, etc.) and leverages the native capabilities of the device. Contextual information often requires access to the native capabilities of the device.

Contextual Information Requires Access to Native Capabilities of the Device

- Develop HTML/ CSS/ JS apps for mobile platforms
  - https://cordova.apache.org
- Run as a native (hybrid) app
  iOS, Android, Windows Phone, Blackberry, and more
  - Native Web View displays the html css/js code
- Interface with native components
  - Accelerometer, Compass, GPS, etc.
- Simple command line interface
  - Run on device (even iOS!) from a single command
    - cordova run —device [ios|android|…]
    - Largely for testing
  - Deployment/packaging from command line
- Easy to use external js libraries (like Diffusion JS Client)

Good app experiences are as much dictated by what data you don't send as much as the data you do.

For some app developers, they want to write code in JavaScript and HTML5 and want fast ROI for the app. They want to be able to get their apps out quickly, but still want to deliver this contextual information. One of the ways you can do that is use technology such as Apache Cordova, a cross-platform technology that gives JavaScript access to native device capabilities.

Data at the Heart of Reactive Apps

If you want to deliver rich, real-time experiences – on mobile platforms – you need to think about data optimization, responding to the network, going reactive in the way that you've developed applications and using contextual information to enrich those reactive experiences.

Good app experiences are as much dictated by what data you don't send as much as the data you do. Some off-the-shelf products and frameworks can take the pain out of development, for example, Apache Cordova and Push Technology's Reappt and Diffusion data distribution solutions.

About Push Technology

We make the Internet work for our mobile-obsessed, everything-connected world. Leading brands like 888 Holdings, DAB Bank, IBM, and William Hill leverage our technology to power applications critical to revenue growth, customer engagement, and business operations. Learn how to deliver apps at scale and speed at **www.pushtechnology.com**

Appendix

Reactive Systems are (as seen in the Reactive Manifesto published on September 16 2014. (v2.0):

Responsive: The system responds in a timely manner if at all possible.Responsiveness is the cornerstone of usability and utility, but more than that, responsiveness means that problems may be detected quickly and dealt with effectively. Responsive systems focus on providing rapid and consistent response times, establishing reliable upper bounds so they deliver a consistent quality of service. This consistent behaviour in turn simplifies error handling, builds end user confidence, and encourages further interaction.

Resilient: The system stays responsive in the face of failure. This applies not only to highly-available, mission critical systems — any system that is not resilient will be unresponsive after a failure. Resilience is achieved by replication, containment, isolation and delegation. Failures are contained within each component, isolating components from each other and thereby ensuring that parts of the system can fail and recover without compromising the system as a whole. Recovery of each component is delegated to another (external) component and high-availability is ensured by replication where necessary. The client of a component is not burdened with handling its failures.

Elastic: The system stays responsive under varying workload. Reactive Systems can react to changes in the input rate by increasing or decreasing the resources allocated to service these inputs. This implies designs that have no contention points or central bottlenecks, resulting in the ability to shard or replicate components and distribute inputs among them. Reactive Systems support predictive, as well as Reactive, scaling algorithms by providing relevant live performance measures. They achieve elasticity in a cost-effective way on commodity hardware and software platforms.

Message Driven: Reactive Systems rely on asynchronous message-passing to establish a boundary between components that ensures loose coupling, isolation, location transparency, and provides the means to delegate errors as messages. Employing explicit message-passing enables load management, elasticity, and flow control by shaping and monitoring the message queues in the system and applying back-pressure when necessary. Location transparent messaging as a means of communication makes it possible for the management of failure to work with the same constructs and semantics across a cluster or within a single host. Non-blocking communication allows recipients to only consume resources while active, leading to less system overhead.