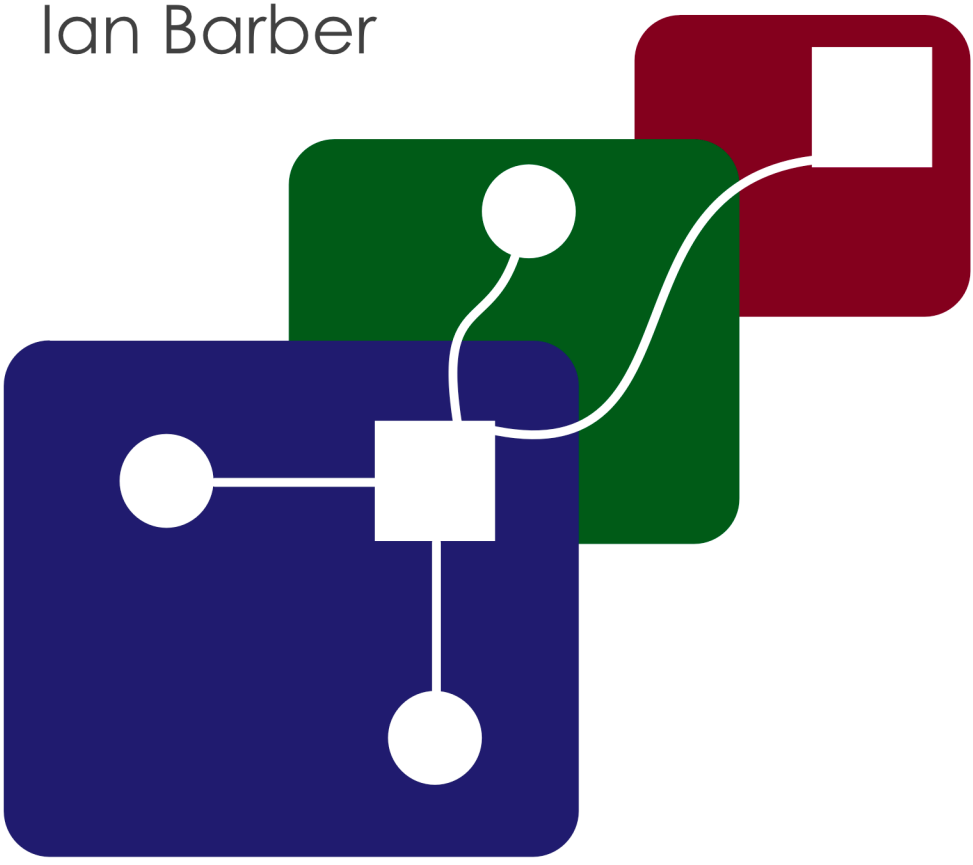
THE MESSAGE IS THE MEDIUM

Ian Barber



The Message Is The Medium

Ian Barber

This book is for sale at
http://leanpub.com/messageisthemedium

This version was published on 2014-03-16

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

Packets On The Wire

It's worth stepping back slightly from the idea of messages to the idea that enabled the networks we have now, the model of *packets*. Almost every network now is built on these ideas, but it took well into the 1980s for the weight of networking to shift. This was a change from the previous view; the telephone companies that had created systems which literally spanned the world by the 1960s had at their heart the concept of a circuit.

This makes sense, as at its simplest a telephone is a device which turns the movements of a microphone into electrical signals on a wire, which on the other end are turned back into sound through movement of a speaker. This was an analog system, because what was transmitted was an analogue of the actual sound - when the air pressure on the microphone was higher, the electrical voltage was higher. To enable *routing* of calls, connecting any caller to any receiver, telephone networks had intermediary devices called *switches* that created a circuit by effectively plugging a wire from one person's phone into the wire for another person's.

To connect different switches trunks of many wires were run between them, allowing the phone company to create a *path* through the network. For a long time, the phone number itself represented this path - from a starting po-

sition, each digit or set of digits became a request to a switch further and further away to connect your circuit to the appropriate line or trunk, eventually finding its way to a point near the location of the receiver where their individual line was selected. Once the path had been found an electrical signal could be shared between two points reliably.

The great problem with this was allocation of resource. By placing a call you requested a two way channel between you and the other party, and that channel was unreservedly yours, even if one or both parties weren't speaking. The capacity of the network was how many calls it could handle at any point, and if there wasn't enough capacity the call simply couldn't be made. Increasing that capacity was primarily a case of adding more lines between two given points on a path.

As the networks became more reliant on automation to perform the switching processes, large computers like IBMs System/360 became popular control points. The QTAM and later TCAM (Queued/Telecommunication Access Method) systems provided a series of simple message queues which could receive and pass on control messages, keeping them in persistent storage. Still though, the voice data flowed up and down the trunks to get from switch to switch with its own dedicated circuit through the telephone network.

Packet Switched Networks

In the early 1960s in the US, Paul Baran was working as a researcher with the RAND corporation, tasked with building a network for radar communications that could survive destruction of various parts of the system. There were multiple lines between the different radar stations and the radar control locations, with their own switches, but because the network formed a tree, though a tree with some thick branches made of many connections, if part of it was destroyed it would cut off the leaves below it. Baran came up with a model that was more like a road network, where there can be multiple routes from one point to another. Then, to take advantage of this he came up with a system that encoded and decoded the information digitally, and chunked the stream of digital information into discrete, fixed size units, called *packets*.

> The term "packet" was actually coined by Donald Davies, a Welsh computer scientist who went on to build the NPL (National Physical Laboratory) network, the first packet network in the UK. Davies independently discovered packet systems while noting the inefficient allocation of access to a mainframe while visiting MIT - each user connected took up an entire phone line, even if they weren't doing very much. Regardless of whether the machine had extra capacity, if all the phone lines were used, no one else

could connect.

By parcelling out streams of data into packets, Baran could take advantage of several benefits, in exchange for having to deal with a certain degree of extra complexity. The model Baran devised was less like a phone call, where all the work was done at the start setting up the path and then allowing data to flow freely down it, and more like the postal service, where the information was parcelled up, and each part placed in an envelope giving the destination. This meant that the packets didn't all have to use the same path, so if part way through sending a stream of data the situation changed (for example a part of the path was destroyed) the packets could take another available route, without either sender or receiver having to be aware of it. It also meant that the whole network didn't have to be aware of exactly where everything was, each switch just had to know the best place to forward a packet on to for a given destination.

The packets could also include error correction information to allow the receiver to detect packets that have been mistransmitted. Via use of a *checksum*, a short code that the sender and receiver could both independently compute from the packet data and check whether they agreed, the receiver could determine the data was bad and re-request or wait for fixed versions if so.

Of course, these benefits didn't come for free. Sending the address each time in the envelope added size to the message being sent. The fact that packets could take different routes could lead to them arriving in a different order than they were sent in, so the receiver would have to put incoming packets into a buffer and ensure they were sorted into the correct order before delivering the data to the receiving application.

However, the really big benefit was the usage of the wire. Rather than having a dedicated path for each conversation, the communication only used as much of the available bandwidth as the size of the data being sent, plus the overhead for the envelope. If one side didn't have any data to send, then it didn't create any packets. This meant that the data from several communications could be *multiplexed* or combined to go down the same physical wire. The switches in-between could control how much was being sent, and if there wasn't enough capacity, they could send packets down another path.

This was a remarkably different way of moving data from one place to another. Where the telephone networks created a link between any incoming wire and any outgoing wire, the switches in packet switched networks followed a *store and forward* model where they would receive a packet from an input, examine it, and determine which was the best output to send it through. This also meant that the structure of the network didn't have to be pre-defined, or known by the data sender - each hop could make best

efforts to move the data towards the destination. This may seem straightforward now, but caused some consternation among telephone engineers for a solid couple of decades.

> We then were able to show that it did not take very long for the self-adaptive behavior to occur efficiently so the network would be able to learn quickly where each node was even though each node had zero information at the start. A byproduct of this phenomenon was that network users' "names" never had to be tied to a physical location. The network could learn where its users were and be able to route traffic efficiently. - Paul Baran[1]

The ARPANET

In response to the success of the project, the US Government created a new body, the Information Processing Techniques Office (IPTO), to extend the work done by Baran and his colleagues. It was tasked with the goal of developing components for standardised, reliable networks - in particular an experimental new network called ARPANET. A number of universities were involved in the collaboration around ARPANET, each of which had their

[1]University of Minnesota Oral History Interviews - http://purl.umn.edu/107101

own computer system and preferred ways of connecting to the network.

This created a new kind of problem - telephone companies had long kept very tight control over what equipment could be put on to their networks, in many places leasing telephone equipment rather than having their consumers purchase it in a store, even for regular phones. Therefore, they could tightly dictate standards that their network equipment should follow. In the ARPANET, everyone wanted to use their own kit, and it all worked differently.

To help with this, Lawrence Roberts at the IPTO took an idea proposed by Wes Clark to create what he called an *Interface Message Processor* or IMP. The idea behind this device was that each team could work on connecting their own computer system to the IMP, while the IMP would speak a single language or *protocol* to other IMPs. This was a practical and brilliant move - by standardising the transportation of data across the network via the IMP, ARPANET could allow different platforms to interoperate without an explosion of mappings between the different systems. The IMPs connecting to other IMPs could route messages flexibly, and implement the adaptive part of the networks, the fault tolerance and multiplexing that made them so powerful.

In 1969, Leonard Kleinrock's team at UCLA sent the first message across the network to Doug Engelbart's group at Stanford - though there was a bug, so the first message technically sent was 'lo' instead of the intended 'login'. This

was quickly followed up with a successful login and the start of the ARPANET as a real, working system.

To support their work, the designers of the IMP documented the protocol it used to communicate. They had to describe the format of each packet sent across the network, how the IMP should respond in various situations, how problems should be handled and so on. This document was released, modestly, as a Request For Comments, which has become the standard way of documenting and standardising protocols on the internet.

> Information is transmitted from HOST to HOST in bundles called messages. A message is any stream of not more than 8080 bits, together with its header. The header is 16 bits and contains the following information:

```
1    Destination     5 bits
2    Link            8 bits
3    Trace           1 bit
4    Spare           2 bits
```

> The destination is the numerical code for the HOST to which the message should be sent. The trace bit signals the IMPs to record status information about the message and send the information back to the NMC (Network Measurement Center, i.e., UCLA). The spare bits

> are unused. - RFC 1[2]

The IMP considered what it transported from one computer to another to be messages. The IMP would then divide the messages into individual packets, and the IMP nearest the receiver would reconstruct the packets into complete messages at the other end. This was a powerful new concept; adding *layers* to networking allowed processes to only have to worry about a limited level of scope without having to understand every detail of the way the communication was facilitated.

In a traditional world, such as that used by the mainframes the telcos used to control their networks, each hop was a complete communication: the sender would send the entire message to an intermediary, the intermediary would send the entire message to the receiver. By using packets, this communication happened on two levels - each packet was a complete communication, but it itself was part of a larger communication - a host would simply send a message to another host, not caring that there were intermediaries who were handling the fragmenting, transmission, and reassembly of individual parts of that message.

Internetworking

Throughout the 60s and 70s there were several other packet networks developed, and there was a strong interesting

[2]RFC 0001 - http://www.ietf.org/rfc/rfc0001

in internetworking between the ongoing research efforts. Taking design ideas from various networks, particularly CYCLADES in France, Vint Cerf, Bob Kahn and others developed a set of protocols: primarily TCP, the Transmission Control Protocol and IP, the Internet Protocol, to facilitate this internetworking.

Based on the learnings from ARPANET and others, some strong fundamental principles were baked into the design of the protocols with regards to dealing with the diversity of networks and clients. Cerf laid down some clear guidance in his initial specifications:

1. Be resistant to failures in intermediate constituent networks and gateways.
2. Be unaffected by the maximum message sizes permitted in constituent networks and by intra- and inter-network timing idiosyncrasies.
3. Be capable of exactly reconstituting a message stream originating in a foreign HOST.
4. Provide for high bandwidth and/or low delay transmission.
5. Require no status information in gateways.
6. Be relatively easy to implement.
7. Permit the receiving HOST to control the flow from the sending HOST.[3]

These principles all have interesting consequences. Being resistant to failures required the ability to resend parts

[3]Vint Cerf, A Partial Specification of an International Transmission Protocol

that were lost, and to handle issues of corrupted packets. Not being limited by maximum message sizes requires the separation of the transmission of the host to host data from the underlying method of transportation, as does the requirement to exactly reconstruct the stream. The lack of status information in gateways means that the status is inferred and that the network would be, in large part, automatically adaptive. The final requirement, that the receiver can control the flow of messages, is a vital point in managing overload and, eventually, congestion.

TCP started out with the same concept of transmitting messages (which would be broken up into packets along the way) from one host to a receiver, but eventually moved to the idea of a *virtual circuit* - that there was a stream of data coming from one host and going to another without a pre-defined endstop. The protocol was split so that IP handled delivery of individual packets, while TCP added on the sequential stream concept, managed reliable delivery and ordering of the stream, and implemented *flow control*, to manage the rate at which packets were sent.

By dividing these two into separate protocols, some beneficial functionality could arise - TCP only had to care about managing reliability, not how to actually route a packet through a network, and IP didn't have to care about reliability at all - it could simply fail if it needed to and rely on the layers above it to resend or respond appropriately.

The eventual system that evolved into the Internet was of a series of networks, communicating via IP between

routers - dedicated IP level communication devices that maintained a table that tracked which output links to push packets to for given sets of end addresses. Later on, similar devices such as *load balancers* would route traffic to specific servers based on higher layer protocols such as HTTP, the web protocol. All this was done to get traffic across this mesh of different services from the network of the consumer to the network of the service they were requesting, and back again.

Distributed Computing

Of course, once multiple machines were available via the network, people wanted to put them to work. One of the first protocols implemented on the ARPANET was RJE, Remote Job Entry. As machines became faster and users wanted results sooner, a style referred to as RPC or *Remote Procedure Call* developed as a method of sharing networked resources. This was first described in RFC 707 in 1976, with Xerox's Courier application and Apollo's Network Computer System influencing many future implementations.

The logic seemed sensible: local programming languages exposed procedures that executed and returned a result, and an RPC extended that facility so that the functionality could be executed on a remote system, and the local code would wait until the result was returned. However, moving beyond one machine added a great many constraints that

were not clearly managed by the procedure call concept - there was latency of communication, competition for access, failing systems, failing networks, congested networks, version incompatibilities, and a host of other issues that made RPC facilities somewhat error prone. RPC coupled two different systems together tightly, and put extra application requirements on stable addresses or name resolution systems - so that the caller could find the system it needed to call.

In response to this many standards were developed: CORBA, the Common Object Request Broker Architecture, XML-RPC, SOAP, DCOM and an alphabet soup of others attempted to remove these barriers, and had remarkable success in many areas. However, the realisation over time was that the more these systems developed, the more they started to approach an idea which was gaining ground in other fields: messaging. While networks quietly, or sometimes loudly, revolutionised much of the world, the idea of messaging is next best looked at in perhaps the field it is most closely associated with, and which has seen some of its most successful use: Finance.

The Information Bus

Financial services have always been one of the most interesting sources of messaging innovation because of the natural fit of their demands to messaging systems. Financial professionals, particularly traders, need to process significant amounts of information, and there is a constant stream of offers and deals being made against the stocks and derivatives available. Dealing with this kind of data coming from different exchanges - where the deals are actually made - and adding in extra pertinent information, such as financial news, means a trader is likely to be interested in a diverse range of updates, which they need access to as quickly as possible. As the markets have become more computerised, moving away from making trades person-to-person on trading floors, and more automated, where trading decisions are made or suggested by algorithms, this need has become even stronger.

In 1975, Vivek Ranadivé came from Mumbai to MIT in the United States to study electrical engineering. He went on to work for Ford, Linkabit (a networking company), and then gained an MBA from the Harvard Business School, where he hit on the idea of developing a way to distribute data differently:

> "My background was as a hardware engineer. If you look inside a computer, there's a back

> plane, or bus, with cards that plug in to run the machine's functions. My idea was to create a software bus and plug applications into that."
> - Vivek Ranadivé[4]

The idea was manifested as TIB - The Information Bus, and at its heart was a *Publish Subscribe* or Pub/Sub system. Most network applications at the time followed a client-server, or *request/response* model where a client would make a request to a server, and a receive a reply. This was limiting in the financial cases in particular because the server could only send information when the client requested it, and clients had to keep track of which servers they needed to access to request different types of data. This worked well for making software that was driven by users doing things, but didn't work so well when software needed to be driven by events happening elsewhere. Several 'push' technologies had been created that were really no more than the client *polling* the server. Polling was just making a request to the server to see whether there had been any changes at a regular interval in the background. This was still somewhat slow to respond, and put a significant load on both the network and the server handling the requests.

In a Pub/Sub system as Ranadivé developed with his company Tibco in the late 80s (originally under the name Teknekron Software Systems, later renamed for "The Infor-

[4]Driving the Information Bus - http://hbswk.hbs.edu/archive/1884.html

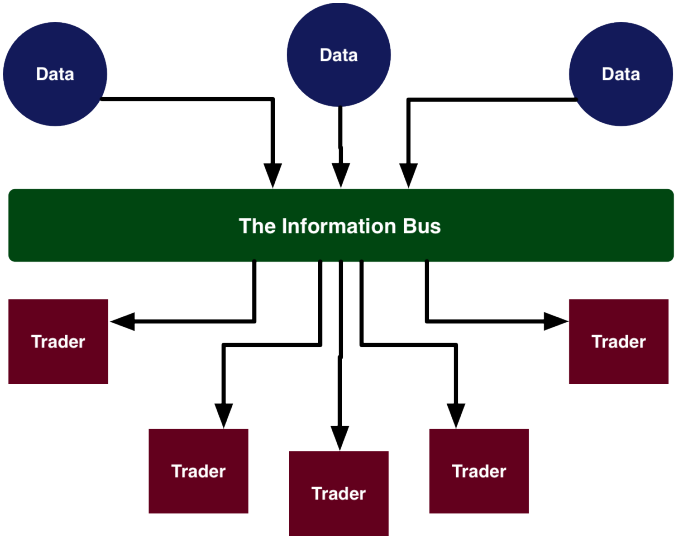
mation Bus Company"), clients become consumers or subscribers, registering their interest in types of information. Publishers produced messages as and when appropriate data became available, and the 'information bus' delivered the messages to all interested consumers.

Subscribers registered their interest against specific *topics*, which were usually strings of characters indicating what type of data was contained within a message. Publishers indicated which topics their messages were covered by. This was a major improvement on the previous model of data delivery, which was more like sending an email to a specific email address. With topics, it worked like a mailing list, where a copy of the same message could be sent to everyone interested in the subject of that list.

Before this system each trader on a trading floor would get a bit of everything - someone who only dealt with tech stocks would still get to see the data about agriculture, for example. By moving to pub sub, traders could get customised views of exactly what they were interested in, by subscribing to specific feeds of data.

This was hugely beneficial in that it simplified the mountain of different systems that financial traders had to deal with. Rather than having each system require its own client or terminal, simple adaptors could be written, much like the ARPANET IMPs, that integrated with financial news and market data systems, processed them into messages and pushed them on to the information bus. It made the coupling between the data sources and the data consumers

very loose, and meant the difficulty of scaling up the number of connections and volumes of data involved was mainly a challenge of scaling the information bus, which was significantly simpler than upgrading each individual producer.



Conceptually, the information bus sat in-between the traders and the data sources

In a traditional network, packets would be routed towards the target system based on the target's IP address, a 32 bit number that is unique to a given host on the network. Routers could learn where to send messages for certain ranges of IP address by publishing and listening for the IP addresses they were aware of and their distance to them

in terms of number of hops. Routing a packet just involved looking up the target IP address in a table of known ranges, and choosing the network port with the shortest distance. If that path became unavailable or congested, the router could pick a different option, and send the packet onwards. The Information Bus worked fundamentally differently. By building a layer on top of the existing network, it could ignore the problems of mapping addresses to routes, and focus on mapping addresses to the topics they were interested in.

The Information Bus servers became additional routers, knowing which clients were interested in which topics, matching the incoming messages to the subscriptions, and sending the messages on to all interested parties. In Pub-/Sub topic based addressing, there is no single endpoint which is being targeted, but an ever changing collection of receivers. In many ways, subscribing to a topic was really a signal to join this group, and on receiving the signal the messaging system could take steps to optimise delivery for the newly enlarged group. Some subsequent systems have used even more abstract routing, such as *content based addressing* where the message is routed to certain receivers based on a complex query that inspects the specific contents of a message - though this generally comes at a significant cost in terms of performance or resources required to push the messages. Pub/Sub took the (established) ideas of message queuing and message passing and made them more powerful by adding more flexible addressing.

> Like many great ideas, messaging had occurred to several engineers over the years, sometimes more than once: "By 1986 we had discovered no less than 17 separate, completely independent examples of yet another group inside DEC "inventing" message queuing! Each independent "inventor" seemed to go through the same process of denial and emotion when they discovered that they were neither the only person nor the first person to come up with the idea for message queuing. I don't know of any greater example of Not Invented Here Syndrome than the message queuing wars that went on inside DEC in the mid '80s!" - Erik Townsend[a]

[a] Erik Townsend, The 25 Year History Of SOA - http://www.eriktownsend.com

Middleware

Tibco's TIB, and later Rendezvous, software was used by a number of banks and financial companies, and extended into other businesses that could benefit from flexible distribution of data. However, they were far from the only provider, and the term *Message Oriented Middleware* started to be used for the stack of software services which distributed the messages. IBM developed their WebsphereMQ product, Microsoft released their entry, MSMQ,

and a host of other companies started selling in to this market.

The middleware, as the name implies, is the software that sits in the middle and actually handles the message distribution. The model defined by Tibco and others was to have services known as *message brokers* receive the messages directly from the producers or publishers, perform any transformations or filtering required, and then route the messages on to subscribers. The broker would usually offer a degree of persistence, so that consumers could come and go and the broker would still try to deliver messages that they might have missed. In this way, brokers provided both message queuing - often one queue per topic, or per consumer, and message passing where they would facilitate the actual delivery of the message from end-to-end. Most brokers would also include various degrees of failure recovery, configurability, audit logging, security filtering and other features needed by the major clients of the time.

Some services, such as IBMs MQSeries or Oracle's Advanced Queue would use a central server to provide the middle layer. The method used by Tibco's RV system, and to degree others such as Talarian's Smart Sockets, was to have many small services running that could provide local queueing for application data, and then communicate with each other - intelligently falling back to alternative brokers in the event of failure, for example.

Having part of the broker on each node meant the system

was distributed, and peer-to-peer. When a node joined, it located nearby nodes and was capable of passing information to others, depending on the relative cost of accessing them. In this way, TIBCO implemented a messaging network which overlaid the TCP or UDP network that it was based on.

This used *multicast* between the services, a topic we'll come to a little later. The downside of the local model was the extra hops between the different nodes, which added latency. For much of the usage at the time, the impact was fairly negligible, only becoming more of an issue later as the volumes started increasing and demands for reaction times below the human scale mounted - again a topic worth revisiting.

Event Driven

Ranadivé's vision was to simplify the lives of the traders who would have to sort through reams of information and make timely decisions, often for significant amounts of money. He saw traders as being on the front lines in a wider organisational change that would see companies move towards being more *event driven*. Normal behaviour would be handled as a matter of course, automatically, and only exceptional circumstances would generate events, communicated through messages, which would register at the desks of everyone that needed to know about them, via the information bus.

Everyone at a company would become the equivalent of a trader, tuning their consumption of knowledge and acting quickly and independently to do their job as effectively as possible. While the impact of the organisational idea is for business management experts to debate, the software idea proved very powerful. Systems which respond to events are an important use case for messaging systems of all types, and have found their way into many, if not most, large infrastructures.