MobilityFirst: An Integration Approach

By Alexander McRae

Introduction

Modern workloads today heavily consist of mobile phones, laptops, and IoT devices, all of which often move network locations requiring intricate work arounds to maintain connections through this network change. Often application developers or system managers are responsible for implementing and organizing the usage of these techniques like the MSocket library and Mobile IP respectively. The reason for these difficulties is a failure of the current internet architecture to handle mobility.

One noticeable issue with how we use the current internet architecture is the conflation of IP addresses and identification. IP addresses are used as routing information, and define a location in the global network where a computer may be addressed. Identities are information which uniquely identifies a computer or thing, an example of this is globally unique identification which uniquely identifies a device. This conflation causes a lot of issues, such as spoofing and hijacking of endpoints. In terms of mobility the conflation makes mobility complicated both for single identity single network location and multihoming, where a single identity maps to multiple network locations.

A project has already been created to handle these issues, MobilityFirst, which starts with a clean slate and builds the internet from the ground up. The project was started in 2010 as part of the National Science Foundation’s Future Internet Architecture Project, which is a group of projects such as Named Data Networking, NEBULA, eXpressive, and ChoiceNet, which aim to look at better internet architectures which suit societies needs more closely. MobilityFirst provides the key components to make a mobile internet and although the team started from a clean slate the components could be used with the existing internet architecture. [3]

The goal of this project is to work off of MobilityFirst and provide operating system level components to modify the *existing* protocols heavily used today and add mobility to them. This paper will cover the components needed both from MobilityFirst and created as part of this project, then, an example of how the components could be used to modify TCP to handle mobility, and an evaluation of the components and TCP modification. Unfortunately, due to time restrictions and a global pandemic, implementations of the components of TCP modification will not be provided.

The resulting solution should have the following properties: Handle all forms of mobility, such as connection time, independent and simultaneous; maintain compatibility with existing internet architectures such as NAT compatibility; be compatible with existing protocols such as IP, TCP, UDP; provide an end to end solution without the need for middleboxes; and provide a general solution which can be used by a wide variety of applications rather than specific applications. Each property ensures that the project has real world usage and could see widespread adoption as well as solving the problem the project aims to resolve.

Background and Related Work

This section of the paper goes through some of the various solutions to the mobility problem used as reference in the final solution presented by this paper. For each solution it is explained how the solution works, what its strengths are, and the reason it is not suitable for the goals of this project. This is not to say that these solutions are not valid, just that they do not meet the requirements set out.

QUIC Protocol [0]

The QUIC protocol sits on top of UDP and provides TCP-like capabilities, as well as other capabilities. Each QUIC connection can have multiple logical connections on top of it which allows for multiplexing across a single connection and prevents head of line blocking. QUIC is used as the underlying protocol for HTTP/3 for this reason.

QUIC handles mobility by identifying a connection with a variable width opaque connection id. Any change in the network location such as a change to IP or Port, which might occur due to a NAT timeout, causes a peer to migrate. As of draft 34 of the protocol only a client may be mobile. Migration is done in the following steps. After the client changes network addresses, it will send non-probing packets (normal packets) from the new address, then a endpoint will do path validation to ensure the new path is usable for the connection, this also includes a challenge to ensure both endpoints are the correct endpoints. Finally the congestion control is reset as it is a new path and flow if migrated over this new path.

This strategy for peer migration suits common use cases well such as the mobile client static server model often found on the web. The protocol also outlines a well defined security model which states what must and may occur during certain events.

The protocol is not suitable for the goals of the project because it does not handle all forms of mobility.

An E2E Approach to Host Mobility [6]

An e2e approach to host mobility is a paper that came out of MIT in 2000. The paper proposes a combination of DNS and modification to TCP to handle mobility. The DNS A (address) record would have a time to live of 0 and whenever a peer moves network locations it would update the record. The modification to the TCP protocol includes an optional header which contains a token. The token is used to identify a connection rather than the original (Client IP, Client Port, Server IP, Server Port). The TCP state machine would be modified to handle this new identification, when a new TCP packet came in it would have to check for the header and IP address to determine if this packet is coming from an existing connection or is a new connection. If the optional header with the token is not present the state machine falls back to normal TCP operation, maintaining compatibility with the original protocol.

This strategy has some nice benefits, it makes use of existing infrastructure such as DNS, it maintains compatibility with the original TCP protocol, and handles all forms of mobility. However, optional TCP headers are often stripped during NAT which is prevalent today, each addressable peer would require a domain name, and the solution is specific to TCP, although a UDP version may be and likely is possible. Overall the paper comes very close to satisfying all the needs of this paper however mobility in the presence of NAT is needed with today's current architecture.

Mobile IP [1]

Mobile IP is a IETF standard which solves the mobility problem by maintaining a consistent IP address through network changes. Mobile IP works by tunneling back to the location the original connection was established. Mobile IP consists of a home agent (HA) and a foriegn agent (FA), the home agent maintains the original connection state and is usually the router the connection was first established on, the foriegn agent is responsible for the tunnel between the mobile host and the home agent. By tunneling all the data back to the home agent the IP address is maintained and the peer connected to is unaware.

This solves all forms of mobility, as both peers can move networks and route back to their home routers without the other peer being aware, maintaining that connection. Mobile IP also maintains compatibility with TCP. Mobile IP does however require large changes to internet architecture such as routers and devices which can support it. The standard is not suitable for the project's goals because it only handles TCP, breaks the end to end principal, and requires large changes to internet architecture (although it has largely been implemented already).

MobilityFirst [3]

MobilityFirst is a project created by the National Science Foundation as part of the Future Internet Architecture Program, where the goal was to give researchers a clean slate and allow them to reimplement the internet from the ground up. MobilityFirst tackles the same problems that are addressed in this paper, except they assume a clean slate.

MobilityFirst handles mobility by decoupling IP addresses and identity and providing different services to help do so. The most important of these components is the Global Name Service (GNS). GNS can be thought of much like DNS except it maps globally unique identifiers (GUIDs) to network locations (IP addresses) rather than domain names to network locations. The GNS has the ability to be rapidly updated. MobilityFirst also provides public private self certifying keys which are used to verify that an endpoint is a GUID. These components are generic so they can be used in different ways by different protocols. The main way these components facilitate mobility is by allowing mobile hosts to update their entry in the GNS rapidly, allowing the peer they are connected to look it up incase of network address changes.

MobilityFirst provides the components to handle all forms of mobility, the potential for an end to end solution, tools for handling NAT, and maintains compatibility with existing internet architectures as the components are generally additive. That being said there is no implementation of any protocols that the author knows of at the time, that maintains compatibility with the original protocol.

MobilityFirst: MSocket [4, 5]

MSocket is MobilityFirst replacement for TCP. The main idea of MSocket is to provide a userspace wrapper around TCP connections. MSocket has its own packet format which keeps separate sequence numbers from TCP. The underlying TCP sockets are known as flowpaths and a MSocket packet can travel over any or all flow paths to the peer. The MSocket connection is independent of the underlying TCP flow paths and as long as there is one valid flowpath the connection persists. When either peer changes network locations they can reconnect 1 of 3 ways, wait for a timeout and use the global name service to reconnect an underlying flowpath, client reconnects to the server assuming the server has not changed network locations, the server tells the client where to reconnect too. This is done for each flowpath independent of each other. Advertising to the peer that you are changing addresses reduces the latency and makes resuming data transfer faster. To facilitate the advertisement connectionless control sockets are used which are effectively UDP sockets which are addressable to the public. These sockets handle the messages and verification of endpoints to ensure data is only being sent to those who should be getting it.

MSocket handles all of the mobility situations, facilitates multihoming, and handles reliability better than TCP does as there are multiple flow paths rather than one. MSocket doesn’t maintain compatibility with TCP (though fallback is possible) which is one of the requirements for the project. Due to this it is likely an application specific protocol rather than a general protocol like TCP and UDP. It also has the issue that it must expose the connectionless control socket to the public, with NATs presence today this is not always simple. However the team proposes a reverse proxy solution to get around this.

Design and Implementation

This section will cover the design of operating system components and services to work with the already established GNS services to help create a basis for implementations of protocols which will allow for mobility. Then a modification of TCP using these services will be proposed to show how they can be used effectively. It should be noted however that when discussing the use of MobilityFirst components and services, the exact usage will stay undefined, that is to say the exact implementations are not referenced, however the main ideas are. For example, in this paper GNS will be defined as any service which reliably maps GUIDs to IP addresses and handles rapid updates to the mapping, not the reference implementation, Auspice.

Mobilityd: The MobilityFirst Daemon

Mobilityd is a daemon which runs very similar to how DNS daemons run on Linux. Mobility's purpose is to handle interacting with the GNS services from MobilityFirst. Mobilityd provides the following functionality:

- Handles GNS lookups, which means it will do the appropriate network requests to retrieve an IP address from GNS server and pass the appropriate GUID
- Handles GNS updates, which means when the device changes network addresses it is updated in the GNS
- Handles self certification, which means when a peer or the GNS challenges the device it returns the correct response to verify the device is the GUID

- Handle peer certification, which means when a peer claims to be a GUID we make the appropriate requests to the peer to ensure that is true

The service should also be accessible by many different ways, like JSON + HTTP and gRPC from external requests, Unix domain sockets and file io from internal requests, and upcalling through the kernel, similar to how DNS works on linux.

API

The API for the service is as follows.

```
fn get_self_guid() -> GUID
```

Returns the GUID of the device, this is used for internal source to advertise to peers and for external sources such as the GNS to have a GUID to challenge.

```
fn get_ip(GUID id) -> Option<Vector<IPAddr>>
```

This API call returns the IP address if possible. If there is no valid mapping to an IP address it returns None. This call will go out to the GNS to get the mapping. It is primarily used internally for the implementation of protocols. It returns a list of IP addresses as GNS supports multihoming.

```
fn get_guid_from_ip(IPAddr ip) -> Option<Vector<GUID>>
```

This API call returns a list of GUIDs which map down to the IP address. If there is no GUID associated with that IP it will return None. It is primarily used internally for the implementation of protocols. Many GUIDs can map down to the same IP address which is the reason it returns a list. Restrictions are put in place to ensure that all GUIDs returned are verified at the IP address so spoofing doesn’t happen by mistake.

```
fn verify_self(Challenge) -> Verification
```

Verify self takes in a challenge from an external source and if the challenge is for the correct GUID responds with the correct result. This is almost always used from external sources to verify the device as the GUID they say they are.

```
fn challenge(IPAddr addr, GUID id) -> Verification
```

Challenge takes in an IP address and GUID and verifies that the IP address is that GUID by calling that device's verify_self api call. The verification returned tells the user whether or not the IP address could be verified.

TCP Mobility

Using the Mobilityd daemon, an implementation of TCP similar to the one found in the end to end approach to host mobility paper can be constructed. First identification for the connection will be established. Whereas before it was (Client IP, Client Port, Server IP, Server Port) it will now be modified to (Client GUID, Server GUID, Token), the token is very similar to connection tokens in QUIC. This means the TCP connection is decoupled from the underlying network routing information (IP address and port). However the requirement of compatibility must be held, so the TCP packet must maintain the specified format. To get around this an optional header will be used containing the token much like the token in the e2e approach to host mobility. Like the e2e paper mentioned, changes to the TCP state machine must be made. In particular the following need to be made:

- On connection if the peer is known to be mobile, which means it has a GUID -> IP mapping (which we can find out either through the mobilityd or optional header token) we switch to using the new TCP version if not we fallback to normal TCP for the time being.
- Onces the peer is established as being mobile we switch to using the new identifier.
- Whenever a new TCP packet comes in we must check if the IP belongs to an GUID mapping or the packet contains a token identifying the connection. If so and we have verified the peer we must route it to the correct TCP application based on the GUID and token.
- If a packet comes in from a new IP address and port combination we must challenge the IP address as the GUID is claimed to be.

Tokens are locally unique, which is to say for any given token there may globally be around TCP connection using that token, however neither peers also use that token. Tokens are also established at connection time with the client setting the token for the connection.

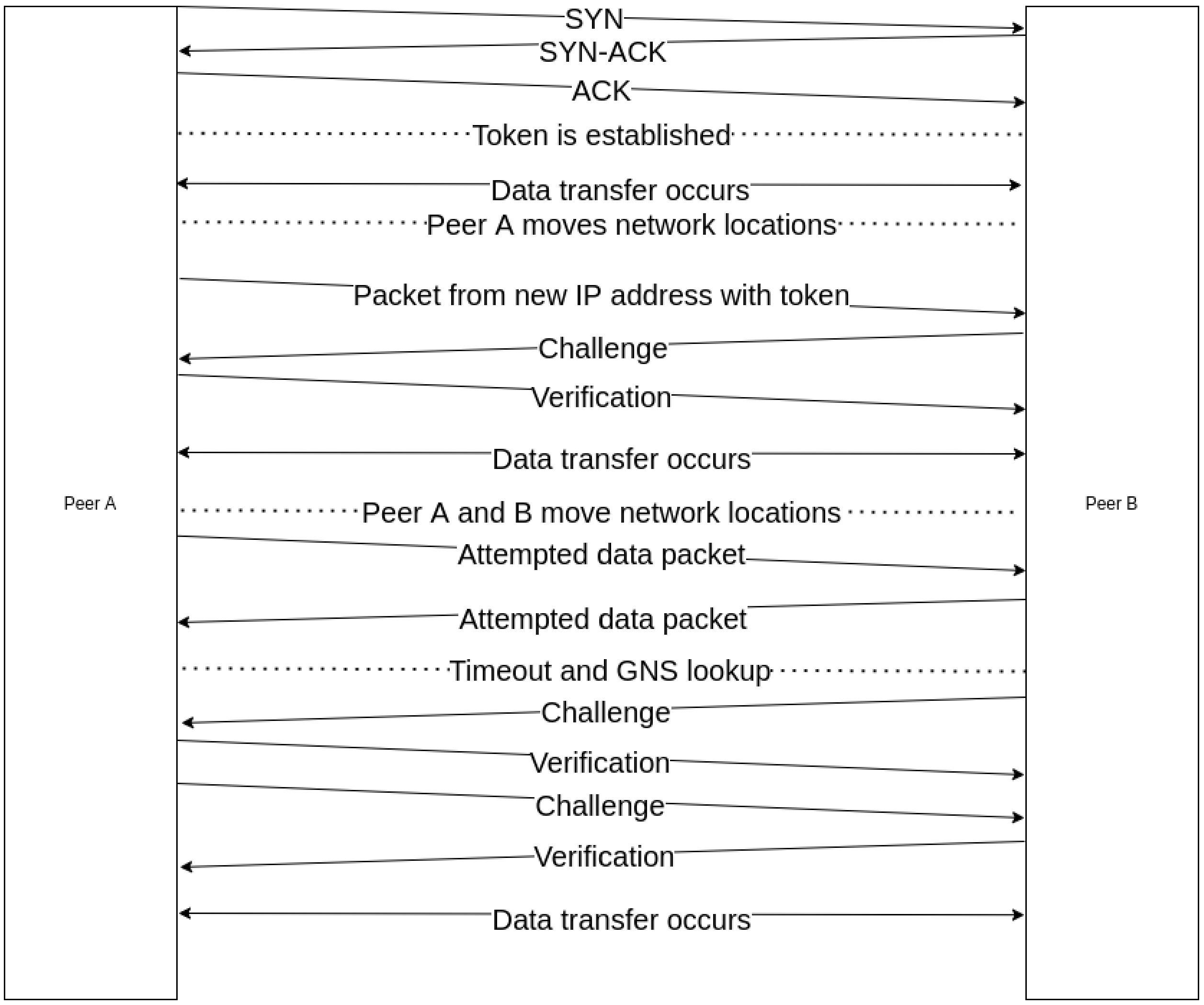
Connection time mobility works by simply reconnecting if a peer moves network addresses as it would with normal TCP.

Single peer migration works as following, first the peer moves network addresses, then it sends an IP packet to the same destination, this will trigger a challenge request from the non mobile peer and allow packets from the new IP address and port to be used on the same connection.

Simultaneous mobility can be handled by calling get_guid_ip from the mobilityd which will requery the GNS to get the updated IP address. A challenge request to that IP address will confirm the identity of the peer and we can maintain the original connection.

Demonstration

Consider a mobile peer A who connects to mobile peer B by looking up the IP address in the GNS.



Evaluation

When looking at the round trip time (RTT) performance of the solution single peer mobility adds an extra 1 RTT and simultaneously requires 2RTT, 1 for the GNS query and 1 for the challenge. This overhead is fairly small and should not cause any performance issues, however real world testing is needed to make sure.

When looking at the overall solutions with reference to the provided criteria the solution succeeds in some areas and fails in others.

Handle all forms of mobility

The solution by using the mobility first components and the modification of TCP provides all forms of mobility. It also does so with minimal overhead. Given this the solution succeeds on this requirement.

Maintain compatibility with existing internet architectures (NAT)

Unfortunately, the solution failed here. The presence of NAT causes issues with how MobilityFirst splits IP and GUID as NAT makes a network address IP + Port rather than just IP. Due to this a token is required for identifying connections rather than (GUID, PORT, GUID, POR) and making the token optional such that it reduces the round trip time. It also makes exposing the mobilityD daemon difficult as NAT hole punching would be needed to expose it. Further, this also causes issues with the usage of the token as often NAT boxes will remove optional headers for security purposes. A proxy, such as the one presented in MSocket, can be used to get around most of these problems.

Compatible with existing protocols (IP, TCP, UDP)

The mobility extension to TCP maintained compatibility with the original TCP stack and could connect to peers without the extension. Given the components proposed and MobilityFirst components there is no reason mobility could not be added to other protocols in a similar manner. Given this, the solution proposed successfully maintains compatibility with existing protocols.

Provide an end to end solution

The solution proposed works with no required middleboxes. However, in the presence of NAT, middleboxes can be used to proxy the connection. Given the prevalence of NAT, a middlebox would be required and further breaks the end to end argument.

Provide a general solution

The solution provided is a drop in extension to TCP, given that TCP is general this extension can be seen as general as well.

Conclusion

Overall the solution is not suitable for a large scale deployment due to the problems will NAT. Although workarounds are possible, they add further complexity, maintenance and difficulty when setting up the solution. Given the evaluation of the project, a different approach should be taken to adding mobility to the internet generally. Given the prevalence of DNS, further work should look to extend DNS with some of MobilityFirsts ideas, such as the splitting of IP and GUID, and extend the DNS daemons to handle this. Although this solution does not solve the issues of NAT it is considerably less infrastructure to setup, integrate, and train people as more are already using or familiar with DNS. If sometime in the future we move away from NAT and restore the end to end principle the solution provided, alongside MobilityFirst provides the correct tools to closely match the uses society has for the internet. It is also quite clear that NAT restricts the mobility available, further work should be done into both mitigating NAT and solutions compatible with NAT but handling all forms of mobility, surely some tradeoffs must be made due to NAT.

References

[0] "DNS Resolver Module¶." *DNS Resolver Module - The Linux Kernel Documentation*, www.kernel.org/doc/html/latest/networking/dns_resolver.html.

[1] "Introduction to Mobile IP." *Cisco*, Cisco, 19 Mar. 2015, www.cisco.com/c/en/us/td/docs/ios/solutions_docs/mobile_ip/mobil_ip.html.

[2] Iyengar, Jana, and Martin Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*, www.ietf.org/archive/id/draft-ietf-quic-transport-34.html.

[3] "MobilityFirst Future Internet Architecture Project." *MobilityFirst FIA Overview*, mobilityfirst.winlab.rutgers.edu/.

[4] MobilityFirst. "MobilityFirst/Msocket." *GitHub*, github.com/MobilityFirst/msocket.

[5] "MSocket Technical Report." *UMASS CS Publications*, web.cs.umass.edu/publication/details.php?id=2326.

[6] Snoeren, Alex C., and Hari Balakrishnan. "An End-to-End Approach to Host Mobility." *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking - MobiCom '00*, 2000, doi:10.1145/345910.345938.