Elisha: Improved Mobile Peer to Peer Connectivity Using Fixed Identifiers

Thaddeus Diamond
Yale University, Dept. of Computer Science
Arthur Watson Hall, 51 Prospect St
New Haven, Connecticut
diamond@cs.yale.edu

ABSTRACT
Many popular applications use persistent sessions to create long-term connections between communicating nodes. However, when both nodes are highly mobile, the likelihood that they become disconnected increases drastically as their session continues. Traditional routing architectures have used a triangular layout with a fixed server to act as a “middle-man” for passing messages between the two mobile nodes. Elisha is a novel, subscribe-publish routing scheme that allows highly mobile peers to connect to each other directly via standard Internet protocols. By removing the need for a fixed server Elisha achieves reduced end-to-end network latency and removes the need for virtualized connections. Moreover, by separating a node’s identity from its location-based IP address, Elisha allows devices to be labeled and tracked, regardless of what access points they connect to the network through.

1. BACKGROUND AND INTRODUCTION
For several years, users have been connecting to the Internet from mobile devices. Unlike traditional fixed computers, which were the only platform for connectivity when many of the Internet protocols were established, these nodes can travel between access points and change the IP addresses that denote their connections.

This method of connectivity has not generally presented a problem, as typical Internet applications delivered payloads over such a short window of time that the device’s IP address would not change. However, recent advents in mobile technology have increased the pervasiveness with which applications requiring long-lasting sessions are used. For example, video communication applications such as Skype are typically used from a mobile phone device over the span of several minutes or hours.

In order to prevent disconnection when a device changes the access point through which they are connected, several protocols have been created to allow a highly mobile node to connect to the Internet for a long duration of time. Virtual Private Networks[10][6] use virtual connections to a designated server to reroute packets such that it appears as if the client is connecting from within a private network regardless of where the actual connection occurs. Mobile IP[11] uses a fixed home agent to connect a mobile node with a foreign Internet host. The home agent uses a virtual tunnel, along with IP layer packet encapsulation, to intercept all transmissions to and from the mobile node. Both of these methods of connecting to the Internet incur serious reductions in throughput due to the requirement that all traffic be forwarded through a centralized server.

Several other schemes have attempted to sidestep the use of a centralized server through the use of multihoming. Multihoming uses the different interfaces available on a mobile device, such as 3G and WiFi, to duplicate network traffic in an attempt to increase reliability and throughput. For example, SCTP[13] uses multihomed endpoints to provide reliable, sequenced stream transmissions to a mobile device. Novel schemes have attempted to layer this protocol on top of Mobile IP[5][1] to achieve greater reliability and throughput. Multipath TCP[12][14][15] attempts to achieve reliability through multihoming, dynamic monitoring and alteration of interface throughput, and a slow-handoff process that occurs when an interface becomes inactive.

However, these schemes are founded on the assumptions that two interfaces are readily available and that only one of the nodes is highly mobile. When two nodes are mobile, as is often the case in current peer-to-peer mobile device applications, the robustness of the previous protocols degrade. Consider the scenario depicted in figure 1. Alice and Bob are communicating over a single association, possibly multihomed. In section (b) of the figure, both Alice and Bob move to new locations at the same time, triggering changes in the interfaces at which they can be reached. Using SCTP, Multipath TCP, or any other multihomed scheme, the association will not be updated in time and Bob and Alice will try to send data to their respective peers’ old IP address, as indicated by the solid unidirectional arrows in section (b). Instead, the two should ideally be sending packets to each others’ new interfaces as indicated by the bidirectional dashed line.
Elisha, the protocol proposed herein, attempts to allow highly mobile peers to create persistent sessions without requiring that all traffic be routed through a fixed server. In order to achieve this goal, the system separates a node’s topological identifier from that node’s identity. This general concept has been proposed several times, including in schemes such as the Host Identity Protocol[9] and the Unmanaged Internet Protocol[4]. In fact, the latter has provided several mechanisms by which the location-independent device identifiers can be implemented to ensure scalability and cryptographically-secure connections[3].

In section 2, we discuss the Elisha protocol in detail and how mobile nodes can use this protocol to connect to mobile peers. Next, we discuss the specifics of the current implementation in section 3. In sections 4 and 5, we discuss anticipated experimental results and the remaining work required of our implementation. Section 6 discusses the project in the context of the Directed Research course through which Elisha was implemented. Finally, in section 7, we conclude.

2. SYSTEM ARCHITECTURE

Elisha uses a centralized server to achieve connectivity over a long-lasting session by using a publish-subscribe model. As can be seen in figure 2, a node, in this case Alice, is located at IP address 1.1.1.2 and wants to connect to Bob, located at 1.1.1.3. We use Alice and Bob to denote logical identifiers that represent two nodes across their entire lifetime.

First, Alice initiates our protocol by performing a lookup in a standard domain name system (DNS). However, instead of returning a static address representing Bob’s location, the DNS returns a static address representing the location of the rendezvous server that is responsible for managing Bob’s location. It is simplest to assume that the DNS described works similarly to the ones in existence today, in which Bob can register his unique identifier at a rendezvous server for some fixed amount of time and, therefore, the rendezvous server need update the DNS listing very infrequently.

After the DNS returns 1.1.1.1, the location of the rendezvous server, Alice then makes a request to the rendezvous server for Bob’s location-based IP address. The rendezvous server at the bottom of figure 2 represents a high-throughput, 24/7, low-latency server that maintains two simple key-value databases. The first maps a set of unique identifiers to the IP address denoting the location of the node represented by that identifier. For example, this table would already contain <Bob, 1.1.1.3> before Alice initiated the protocol. The second table contains a set of pairs that establish subscriber-publisher relationships. The rendezvous server returns 1.1.1.3 to Alice and, in addition, adds the subscriber-publisher relationship <Alice -> Bob> to its database. This representation lets the rendezvous server know that Alice is about to begin a session with Bob and, should his location ever change, she should be updated with his new IP address.

Finally, Alice has Bob’s temporary mobile IP address, 1.1.1.3, and uses a standard Internet protocol such as UDP to make a direct connection to him. The two then exchange messages as in any typical connection.
2.1 Introducing Mobility

The major contribution Elisha makes to connectivity is the ability for nodes to be extremely mobile, yet remain connected without the use of a triangular routing scheme. Let us assume that Alice and Bob have begun transmitting to each other after using the protocol established in figure 2.

Now, assume that Bob moves his location, changing the IP address at which he is available. As soon as this occurs, the Elisha client, running on Bob’s device, detects that the device’s interface has changed and sends an update to the rendezvous server containing the new IP address as shown in figure 3(c).

Upon receiving Bob’s new IP address from the Elisha client, the rendezvous server responsible for Bob publishes that update to all registered subscribers (not pictured). Alice, being among these subscribers, receives that update through the Elisha software running on her device, as shown in section (d). In the meantime, the application Alice is running continues to transmit packets, but does not receive any acknowledgements of receipt from Bob. These packets are now lost, as no one was available at Bob’s old IP address to receive and process them. However, the Elisha client, which monitors all messages sent out to the peer during the session, automatically retransmits all packets for which it had not received an ACK. These correspond to the dashed lines shown in section (b), (c), and (d) of figure 3. As shown in section (f), Bob then sends back the acknowledgment that the intermediate packets were received. Finally, the session resumes without loss (not pictured).

It is important to note that the rendezvous server must retrieve Alice’s latest topological identifier from the rendezvous server responsible for maintaining that information. However, due to timing, Alice may not receive the broadcast on the first attempt. If Alice moves her location at the same time Bob does, but the rendezvous server responsible for maintaining Bob’s location requests an IP address to broadcast to before Alice has updated her respective rendezvous server, the broadcast will be lost. This issue can be solved in one of two ways. First, and most simply, the rendezvous server could retransmit all broadcasts using an exponential backoff until it receives acknowledgement that the broadcast was received. However, using the unique identification scheme briefly discussed in section 2.3, initially proposed at length in [3], these redundant broadcasts can be avoided altogether. Assume that every message is digitally signed with the transmitting node’s unique identifier. Because the broadcast is not sent out until after the update from a mobile node has been received and logged, at least one peer, say Alice, will receive an update even if both move at the same time. Upon receiving this update, Alice could send Bob a dig-
Figure 3: Maintaining Connectivity During Mobile Session
2.2 Rendezvous Servers

As was stated previously, it is easiest to think of rendezvous servers as being analogous to telephone carriers. An individual registers a fixed identifier (i.e., his/her telephone number), with a single rendezvous server (i.e., a mobile phone provider). The rendezvous server is then responsible for providing the top-level DNS with a list of fixed identifiers that it has registered in its database, along with each one’s respective time to live.

A rendezvous server must have several desired properties in order to be useful in the Elisha protocol. First and foremost, since it is sent requests at the beginning of every peer-to-peer session, it must be highly available. Moreover, it must be able to perform extremely fast associative lookups in order to maintain pace with the rate at which requests for topological identifiers are arriving. Finally, it must ensure periodic redundancy so that, should the server need to be taken offline or endure an attack, the mobile devices need not register with that node again. Many of these desired properties have been extensively investigated in the context of highly available, replicated, and fault-tolerant web servers[7][2][8], and could be repurposed and optimized for use in this protocol.

Interesting variations of the simple server design are possible. For example, rendezvous servers could operate in a hierarchical fashion. In such a scheme, an individual would register with a top-level rendezvous server, which in turn would have several smaller delegates at various locations. When a peer attempts to establish a connection to that individual, it would execute the protocol as usual. However, instead of receiving updates from the top-level rendezvous server, it would receive updates from a delegate that was closest to the individual at the time of the connection.

This hierarchical composition of rendezvous servers is vital to reducing latency for nodes that travel long distances. Assume a United States citizen registers with a top-level rendezvous server in New York. That top-level server has delegates in New York, France, and China. After traveling to Germany, a German citizen running Elisha initiates a connection with the US citizen. In this case, it does not make sense for the US citizen to send updates about its location back to New York, as the network latency would severely disrupt the application’s ability to function smoothly. Instead, as soon as the subscription request arrives for the US citizen, the top-level rendezvous server would send a notification to the US citizen’s Elisha daemon with the IP address of the delegate in France. In addition, a notification with the new publisher-subscriber pair would be sent to the server in France. Then, for the duration of that session, the US citizen’s device would send updates to the server in France, which would act as a full rendezvous server for the protocol.

It is important to note that, barring highly unusual circumstances, it is only necessary to assign a delegate at the onset of a new publish-subscribe request. This is due to the assumption we make that the duration of a session is not long enough to cross sufficiently large geographical areas.

2.3 Unique Identification

An extremely important implication of Elisha is the ability to label nodes with permanent, unique identifiers. Intuitively, these could be thought of as simple indicators much like telephone numbers or domain names. As is the case today, the DNS would be responsible for managing the delegation of those identifiers, as well as managing which rendezvous servers are responsible for which client.

However, what is perhaps more interesting is the possible use of these identifiers as cryptographic primitives. Consider the case where an individual’s unique identifier actually represents that person’s public key in a designated asymmetric cryptosystem. In that case, the mechanism by which mobile nodes update the rendezvous server with their new address could be cryptographically secured, preventing spoofing attacks and reducing the complexity necessary for the protocol’s trust model. Furthermore, any pair of nodes could use the identity of the node with which they are attempting to communicate as a digital signature for all messages sent to that node, obfuscating the need to obtain a person’s public key via other channels.

3. IMPLEMENTATION

Elisha was implemented using C++, is under revision control using Git, and has a set of documentation associated with it generated automatically by Doxygen. The actual source repository is available on GitHub, along with the most up-to-date documentation. Currently, the package only supports applications that connect over UDP using IPv4 addresses.

In order to provide a complete demonstration, a simple DNS was implemented. This was an extremely crude approximation that initialized a set of unique identifiers a priori and their corresponding rendezvous servers. Therefore, the demonstration only works when using unique identifiers that are contained in this set.

The rendezvous server’s implementation is slightly more sophisticated, providing dynamic registrations for any arbitrary unique identifier, as well as providing real-time updates to all subscribers of a given node when
that node’s location-based IP address changes.

The client-side daemon of Elisha was run as an object in a separate thread. Figure 4 shows a complete FSM representing the client software’s main operating loop. The object, which could be configured to communicate with an arbitrary rendezvous server, continually polled the device’s interface and, if it ever detected a change, would dispatch a message via UDP to the rendezvous server. Moreover, if it ever received a message from the peer’s rendezvous server, which is known ahead of time for any session, it would automatically retransmit the set of network messages that had not received an ACK. These network messages were represented as std::strings.

3.1 Application Development

In the paradigm developed, applications are developed on top of Elisha’s client-side software package. These applications still use the standard methods of Internet connectivity (UDP, TCP, SCTP), but must make several calls to the Elisha API in order to utilize the protocol.

Figure 5 shows pseudocode for a sample application that sends a single message to its peer and then prints out a message when it receives acknowledgement that the message was received. First, the application must start an Elisha daemon that can run in the background. It is important that this daemon be started in a separate thread so that it can continually monitor available interfaces and detect changes, and poll the open connections for updates about peers’ locations.

After the mobile node has been started, the application receives an address to which it can send the application data by asking the Elisha daemon to connect to that peer using the standard protocol. In the case that the protocol fails at some point during the stages depicted in figure 2, the Elisha API call will return NULL. It is necessary then for the application to loop until the peer struct is appropriately filled or exit.

The application then proceeds to send a message to the peer and, after doing so, alerts the Elisha daemon that the specified message has been sent. This allows the daemon to keep a buffer of dangling messages, which, in the case of a disconnection, can be retransmitted to the peer automatically.

Finally, the application blocks until it hears back from the peer. Once it does, it alerts the Elisha daemon that it has received acknowledgement so that the message can be freed from the buffer pool, and then it repeats the process.

3.2 Native Linkage

In future iterations, Elisha will be linked natively using compiled object files and public interfaces. Unfortunately, due to time constraints this was not possible for this iteration. However, it is a high priority on the list of features to be implemented in forthcoming versions of the software.

Many advantages can be conferred by implementing Elisha as a native library. First and foremost, a reduction in the amount of code required would be highly advantageous. In its current implementation, an application must not only notify Elisha that it is about to send a message, but it must also send that message.
using an existing Internet protocol. Moreover, because actually sending the message across the network is decoupled from notifying Elisha that a message is being sent out, fewer statistics can be collected and monitored. These metrics would allow the daemon to make runtime decisions that could boost performance and reduce unnecessary overhead.

4. ANTICIPATED RESULTS

Unfortunately, due to time constraints, only a demonstration was able to be implemented. Ideally, there would have been a set of experiments conducted using portable devices to measure the performance over a long period of time. However, this was not possible during the course of this research.

We would expect several network connection characteristics to be different when running Elisha versus a standard triangular route, such as Mobile IP. First and foremost, the latency in messages between nodes would be approximately half as long. As can be seen in figure 6, during a session using Mobile IP packets need to travel from one peer to a designated hop, the home agent, and then finally to the peer. Elisha, which uses a direct connection to the peer to send data, does not incur this performance reduction.

It would be easy to theorize that perhaps the average latency over the course of the entire session might approach levels consistent with those found in Mobile IP due to the lengthy process that occurs in Elisha when any node changes location. However, analytically, both Elisha and Mobile IP handle the changing of a device’s interface with two network messages. In Elisha, this corresponds to an update from the device when its interface has changed, along with a broadcast from the rendezvous server to all of that device’s subscribers. In Mobile IP, this corresponds to the process of establishing a new virtual tunnel to the home agent so that the home agent does not forward packets to the device’s old location. Therefore, assuming both usage cases had the same rate of mobility, they would most likely experience roughly the same delays in connectivity when the device changed location.

Furthermore, it is possible that Elisha might in fact achieve a higher total throughput than Mobile IP. In Mobile IP, the total throughput of the connection is limited by the ability of the home agent to forward incoming and outgoing packets. However, Elisha does not rely on the characteristics of any external server. Rather, Elisha uses standard Internet protocols to fully utilize the current Internet infrastructure’s incredible elasticity. Moreover, were the routing protocol used by communicating peers to support multihoming, the transmission rate would be further augmented. Multihoming in Mobile IP could, in fact, reduce the throughput, as the home agent would need to process and forward more packets.

4.1 Sample Application

In order to provide a working sample of the system, an application was created that utilized the client-side daemon referenced in section 3. This application was a simple “echo chamber.” A node would send out a one-word message to the designated peer and would wait for a response before sending out another one. Whenever

```
1: start elisha_daemon
// Any time we connect to a new peer, we make a call to the daemon
// which implements the protocol
2: begin do
3: peer = elisha_daemon.connect_to_peer(peer_logical_address)
4: while peer is null

// The daemon needs to be alerted when a message is sent so it
// can resend in the case of a disconnection
3: print "Sending message to peer..."
4: peer.send_message(message)
5: elisha_daemon.message_sent(message)

// When an ACK is received the daemon is alerted so the data
// is not retransmitted on failure
6: ack = peer.receive_ack_blocking
7: elisha_daemon.ack_received(ack)
8: print "Received acknowledgement that the message was sent!"

// Send another message
9: goto 3
```
the node itself received a communication it would echo it back to the peer who sent it. In such a setup, two communicating devices send each other two messages and receive two replies every second.

In our implementation a node does not send another heartbeat until it receives a response. The response serves, essentially, as an ACK so that we can follow the application paradigm referenced in 5 without having to monitor the low-level statistics of the socket. If a peer changes location, the Elisha daemon automatically resends the most recent heartbeat, since the peer would not have received it.

This application was extremely useful in demonstrating the system for many reasons. First and foremost, the use of a heartbeat creates a controlled, long-lasting session in which, once a second, each node sends and receives two messages. Any deviation from this number is noted as an irregularity and provides an extremely easy way to monitor session traffic. Moreover, the use of a “text-only” application provides simple debugging that layering Elisha on top of, say, a browser, could not.

5. FUTURE WORK

Several aspects of Elisha will need to be continually improved in order for it to be a viable alternative to existing routing protocols. First and foremost, an accurate and detailed set of experiments must be run detailing the performance of Elisha as compared to existing implementations of triangular routing including Mobile IP. These experiments, as detailed in section 4, need to use more robust and realistic implementations of DNS and rendezvous servers, including the ability to concurrently handle several hundred requests per second.

Furthermore, the use of unique identifiers as cryptographic primitives must be investigated. Currently, the mechanism by which a mobile node updates the rendezvous server is completely unsecured, an obvious security flaw. The use of a simple asymmetric cryptosystem such as RSA would provide essential data regarding feasibility.

In addition, the layering of a multihoming Internet protocol, such as multipath TCP or SCTP, on top of Elisha could provide additional performance boosts. Many mobile devices currently have two interfaces, namely for 3G cellular data and WiFi, only one of which is changing frequently. During the times at which this interface, say WiFi, is changing, the multihoming protocol could be induced to favor the stable interface while the Elisha client updates the rendezvous server.

At the implementation level, the current codebase must be updated to support TCP and SCTP connections, as well as IPv6 IP addresses. As discussed in section 3 only UDP connections using IPv4 addressing are supported. Moreover, as discussed in section 3.2, Elisha needs to be implemented as a separate link-able library so that the API calls necessary in figure 5 can be reduced. Several other items that remain to be completed can be found in the Doxygen documentation under the “Todo List” that it generates.

6. RESEARCH EXPERIENCES

Overall, this course of individual study was extremely rewarding. Although arduous, the ability to be brought onto a project at the point of conception and see it through to the implementation of a proof of concept was exciting. Moreover, I feel that, as an undergraduate working under the tutelage of several professors, it can be extremely intimidating to present one’s insights and attempt to persuade the professors that they are correct. However, being forced to do so in an environment unlike the typical classroom setting allowed me to grow tremendously and feel far more confident in preparing my arguments.

6.1 Code Maintenance

The current version of the software is licensed under the standard MIT open source license and is tracked under version control via a GitHub repository located at https://github.com/thaddeusdiamond/Permanent-IP-With-Mobility. Being able to contribute to the open source community was an exciting experience, and allowed me to learn a great deal about the available open source tools, including Git and Doxygen, that are available for large-scale software development. In addition to the actual source code repository, the most up-to-date documentation about the project can be found at http://thaddeusdiamond.github.com/Elisha-Documentation/

6.2 Issues Encountered

Several obstacles hindered the implementation of this project. First and foremost, due to a misdirection in the initial planning of the system, an entire proof of concept had to be scrapped and started from scratch. Initially, the architecture relied on the adaptation of the architecture to existing software that implemented traditional triangular routing, shown in figure 6, so that the amount of work required in putting together a functional prototype would be reduced. Ironically, this codebase, which contained over 1500 lines of code, consumed at least three weeks of the development process that should have been spent designing and writing the current implementation.

Furthermore, it was only towards the end of the semester that I began to understand the importance of generating automatic documentation with the use of Doxygen, the creation of unit tests from GoogleTest’s unit test framework, and the need to modularize and separate the components of the system. This was not the result of poor planning, but rather a lack of knowl-
edge on my part about the proper methodology for implementing large software projects. These portions were so important that I did, in fact, take the time to rewrite large sections of the codebase to include unit tests and accurate documentation. As a result, I feel much more confident in the ability of others to be able to join the project.

7. CONCLUSION

We have presented a novel Internet routing protocol that allows highly mobile peers to establish direct, persistent connections to each other without the need to use triangular routing. By separating a node’s unique identifier from its topological identity, Elisha provides applications that use persistent connections with a mechanism by which peer-to-peer connections can be easily established and maintained. Furthermore, we have provided a sample implementation that allows mobile application developers an easy mechanism through which to use this protocol.

The future of mobile computing is, as always, continually evolving and uncertain. However, if market estimates are conservatively trusted, the majority of the next one billion people to connect to the Internet will actually be connecting through their mobile devices. This radical departure from traditional models of Internet use requires an equally radical and novel paradigm shift that allows these nodes to connect quickly and efficiently. The Elisha protocol aims to contribute to that radical departure by introducing a new, reactive, event-based paradigm that allows mobile devices to adapt to changes in location by generating and responding to arbitrary signals rather than mimicking or artificially creating a traditional, fixed model of connectivity.

 Personally, this course was an excellent experience, and I would highly recommend conducting Independent Research to any of my undergraduate peers. I discussed it briefly in section 6, but it bears repeating: being forced to work independently on cutting-edge technologies for which there was very little documentation of existing tools and absolutely no standard solution set forced me to mature and excel in a way that I have never been able to confined within a classroom. This experience, among other research experiences I have had, is a major reason that I will be pursuing a graduate degree in Computer Science after I graduate.

8. ATTRIBUTION AND THANKS

Thanks is due to Professor Yang Richard Yang, Professor Bryan Ford, Professor Avi Silberschatz, and Ramakrishna Gummadi for their continual support and help in completing this research project. Also, thanks and credit to Alexander Thomson and Shu-Chun Weng for bits of code borrowed from reused Makefiles.

9. REFERENCES


