Enabling Renewed Innovation in TCP by Establishing an Isolation Boundary

Umar Kalim[‡+] Eric Brown[+] Mark Gardner[+] Wu-chun Feng[‡]
Department of Computer Science, Virginia Tech[‡]
Office of Information Technology, Virginia Tech[+]
{umar, brownej, mkg, wfeng}@vt.edu

ABSTRACT

The growth of the Internet has ushered in and established the "Information Age." However, its success has also arguably increased the difficulty of incorporating innovative changes that are needed to develop further functionality for next-generation networked applications. From the transport perspective, the desired functionality includes (1) supporting multiple network paths, (2) providing transport over hybrid networks (e.g., using both packet- and circuit-switched networks), and (3) decoupling upper-layer services from endpoint-naming semantics. The need for functionality — such as transport composability — has been reiterated in recent research and leads to an apparent dilemma: TCP, the ubiquitous transport protocol, neither admits such functionality in its present form nor does it seem possible to add it without substantial modifications. Furthermore, radical changes — whether through incompatible extensions or by creating a completely new protocol — will not be easily accepted.

In contrast to the apparent dilemma, we argue that a backward-compatible modification to TCP that supports increased functionality is possible without incurring significant burden in additional protocol exchange. The lightweight mechanism, built upon a set of TCP options, establishes an *isolation boundary* between TCP and the application. The boundary separates an application data stream from the TCP transport flow. Further, it provides for the establishment of a control channel that allows additional capabilities to be negotiated dynamically throughout the lifetime of the communication. In short, the mechanism provides a simple "hook" into TCP with which new features can be realized. This increases the freedom to evolve TCP while maintaining compatibility, thereby facilitating incremental adoption.

Categories and Subject Descriptors

C.2.2 [**Computer Systems Organization**]: Computer-Communication Networks—*Network Protocols*

Keywords

Next Generation Networks, Transport Layer, Specification.

1. INTRODUCTION

Underneath virtually all networked applications, the Transmission Control Protocol (TCP) busily delivers packets of data. At our university, for example, TCP delivers over 2 TB per day just over the campus wireless network. Clearly, TCP has been wildly successful. Its simplicity and ease of implementation has resulted in wide adoption, and it has materially contributed to the growth of the Internet [1].

Along with ubiquity comes increased expectations. Where users were once content to simply transfer files between machines, now they expect the network to provide functionality that TCP was never designed to support. For example, users now expect the network to be seamlessly available on whatever device they carry, wherever they carry it, and to transparently switch between networks as they roam. They want to take advantage of all available networks simultaneously for increased performance or reliability.

The need for greater functionality in TCP has been reiterated in recent research [2–15]. However, TCP neither supports such functionality nor does it appear that it can be modified to support the new functionality in a backwardly compatible way [4, 6, 7, 9, 11]. In the belief that radical changes are required to extend its functionality, some researchers advocate a clean-slate approach as the only path forward [16–19].

TCP has been so successful that change will not be easily accepted. End users and network operators would bear the majority of the burden of transitioning to a different protocol, including assuming unknown risks; and although users demand increased capabilities, they do not want such change. Yet some change is necessary if increased functionality is to be realized.

We observe that application developers build extensions to TCP as part of the application in an attempt to tackle these challenges (e.g., transport composability proposed by Kissel [5] and Habib [9]). This leads to a duplication of effort. Alternatively we see proposals where extensions [8, 11] require applications to use custom libraries to exploit maximum benefit from underlying services.

We argue that by the insertion of a simple "hook," TCP can be made significantly more extensible. Furthermore, it is possible to do so in a backwards-compatible way such that adoption can be incremental.

We build upon the work of Ford [6] and Iyengar [3] and introduce a limited *isolation boundary* between TCP and the application. The purpose of the limited isolation boundary is to decouple an application's data stream from the underlying TCP transport flow to allow protocol designers freedom to extend TCP to implement new functionalities. The lightweight mechanism utilizes a set of TCP options, referred to as the *Isolation Boundary Options (IBOs)*, during the connection setup phase, and provides for the creation of a control channel that endpoints can use to negotiate additional functionality, as appropriate, during the lifetime of

the connection. Since the presence or absence of the new options at connection setup time indicates whether or not a stack implements the extension mechanism, adoption can be incremental. Connection setup falls back to legacy behavior if either stack fails to recognize the new options.

The remainder of the paper is organized as follows: Section 2 discusses related work. Section 3 presents the proposed modification to TCP, discusses how it maintains backwards compatibility while enabling renewed innovation, outlines how new capabilities can be implemented using the mechanism, and ends with a critical analysis of the approach. We conclude in Section 5 with suggestions for further work.

2. RELATED WORK

Network researchers have identified challenges with the TCP specification that inhibit our ability to evolve the protocol stack to allow richer functionality. Among prominent efforts are those by Kissel and Brown [5, 7], Ford [8], Habib [9], Mahieu [10], Salz [11], Snoeren [12, 13, 20], Landfeldt [14] and Maltz [15]; the most recent being that of Iyengar [3] and Ford [2, 4, 6]. Much of the related research addresses individual problems that need to be solved; be it providing support for fault tolerance using multiple network paths (e.g., Salz [11]), providing support for mobility (e.g., Snoeren [12]), developing abstractions to decouple the binding of transport endpoints[1] to entity names (e.g., Ford [6]), or employing diverse transports longitudinally over the Internet path (e.g., Brown [7]). In addition, all the above work indicates that significant modifications to TCP are required (e.g., modifications to the sockets API which may break legacy applications). Others have argued that it is imperative that a clean-slate approach be adopted [16–19].

The papers most closely related to ours are [2–4, 6]. Ford et al. [2] suggest using TCP options to compose an application stream from multiple transport flows. These transport flows may then be mapped onto different network paths (if available). Iyengar et al. [3, 4, 6] discuss a logical refactoring of the transport layer to form the *semantic*, *isolation*, *flow*, and *endpoint* sub-layers in an architecture they refer to as Transport Next Generation (T*ng*), as shown in Figure 1. The refactoring highlights the fact that over time a variety of roles have been coalesced into the transport layer. These roles may be broadly classified as follows: the identification of transport endpoint; performance-related functions such as congestion control; mapping of transport endpoint to entity; and end-to-end semantic functions such as data ordering.

Our proposed modifications are between the transport and application layers in the TCP stack, corresponding to the boundary between the flow and isolation sub-layers in the T*ng* architecture. The isolation boundary decouples the application data stream from the TCP flow in a backward-compatible manner. This decoupling, along with the setup of a control channel, paves the way for substantial extensions to TCP. While this work does not address any of the aforementioned individual problems directly, it creates a framework upon which solutions can be composed.

Examples of these extensions may be composing an application stream from multiple transport flows or setting up a hybrid transport along the Internet path. We maintain that the isolation boundary defined in this paper is a suitable mechanism upon which to build the isolation sub-layer in T*ng*.

[1]By transport endpoint, we mean socket pair (i.e., 4-tuple).

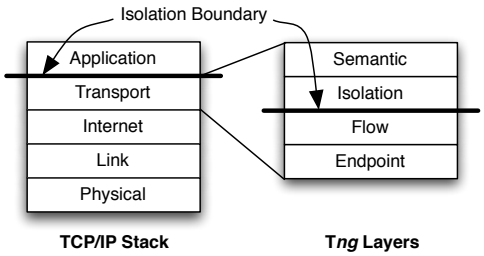

Figure 1: The Isolation Boundary in the Context of the TCP/IP Stack and the T*ng* Layers.

3. PROPOSED SOLUTION

We propose to extend TCP via a set of TCP options, called Isolation Boundary Options (IBOs), to provide a flexible and dynamic mechanism for creating a larger class of extensions.

The IBOs are a "hook" for introducing future extensions. Specifically, the IBOs serve two purposes: (1) decouple the application data stream from the TCP flow that provides transport by creating a logical transport-independent flow that is mapped onto the transport-dependent (TCP) flow, (2) establish a control channel for composing mappings between application data streams and the transport-independent flows in a much more flexible and dynamic way than provided by TCP options. Ways in which the mechanism can be used to implement additional functionality will be discussed in Section 3.1.4, but first we consider the semantics of the isolation boundary.

3.1 Concept and Semantics

We define an IBO to contain two pieces of information: (1) an ID to identify a transport-independent logical flow, and (2) a sequence number from an appropriate sequence space. The ID, denoted the *Transport-Independent Flow ID (TIFID)*, is unique in the context of the participating stacks.[2] As with TCP, sequence numbers orients a protocol data unit in the application data stream. They are also used to acknowledge data that has been received. Sequence numbers used for the former purpose are called *Transport-Independent Sequence Numbers (TISeq)* and those used for the latter are called *Transport-Independent Acknowledgement Numbers (TIAck)*.

TCP stacks advertise that they implement the isolation boundary by specifying an IBO during connection setup. If both hosts specify an IBO, then the isolation boundary functionality is enabled. Otherwise, both fall back to legacy TCP. In this way, backward compatibility is maintained, and there is no requirement that all hosts be updated simultaneously.

3.1.1 Transport-Independent Flow Setup

Consider the sequence diagram shown in Figure 2 for $Peer_A$ and $Peer_B$. A TIFID unique to both stacks is needed to identify the logical flow. One approach is to allow each stack

[2]TIFIDs are not session IDs per se. A session would consist of a composition of transport-independent logical flows and would have its own logical identity. Indeed, sessions are an example of additional functionality that can be implemented with the proposed mechanism.

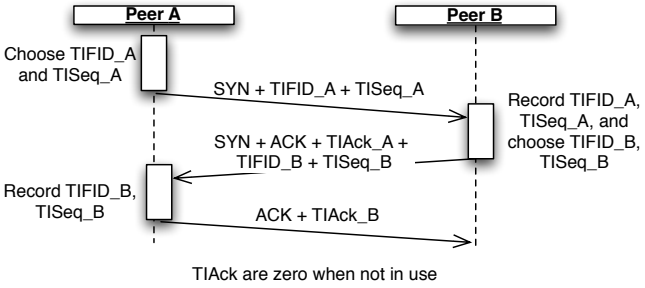

Figure 2: Sequence Diagram of the Exchange of Isolation Boundary Options During Connection Setup.

to select one half of the TIFID. In this case, $Peer_A$ defines the first half of the TIFID using a random value. It also initializes the TISeq number, using a random value, to define its transport-independent sequence space and establishes a mapping between the TISeq and the TCP sequence number. These partial TIFID and initial TISeq are sent to $Peer_B$ in the SYN packet containing an isolation boundary option.

Upon receipt of the SYN packet, $Peer_B$ defines the second half of the TIFID using a random value and also defines its TISeq number using a random value to establish its transport-independent sequence space. Finally, it sends the completed TIFID and its TISeq back in the SYN+ACK TCP header, making sure to acknowledge the TISeq it received from $Peer_A$ using the TIAck field.

Upon receipt of the reply, $Peer_A$ notes the completed TIFID, which uniquely identifies the flow. It returns an ACK packet as the final phase of the three-way handshake, making sure that it acknowledges the SYN it received using a TIAck.[3] At this point, transport-independent flows in each direction have been established, along with the associated bidirectional TCP connections. The transport-independent flows constitute a control channel through which the two stacks are able to negotiate and coordinate additional functionality in TCP.

3.1.2 Transport-Independent Flow Close

Next, we consider how to close a transport-independent flow and release resources. The primary mechanism for a graceful close is a message exchange on the control channel. Alternatively, a graceful close of the underlying TCP connection, i.e., in response to a FIN packet, causes the state associated with the transport-independent flow to be cleaned up. Ultimately, as with any protocol operating over an unreliable communications channel, the use of protocol timeouts are unavoidable. A timeout triggered within TCP would propogate up to the transport-independent flow and either cause it to close or cause it to attempt a reconnection.

3.1.3 Re-Synchronization

At any time during the life cycle of the transport-independent flow, the connection may be re-synchronized by exchanging IBOs in a new TCP three-way handshake. Typically, IBOs are exchanged again when flows resume operation after a TCP disconnection or when communication becomes impossible due to an address change. After re-synchronization, both stacks may safely discard state associated with the old TCP connection. Because the data structures were created when the isolation boundary was set up during the original three-way handshake, the transport-independent flows can be synchronized to a new TCP connection. The procedure for resuming operation after disconnection is the same as for creating a new connection except that the previously completed TIFID is used instead. Since the TIFID is already complete, the receiving stack looks up the isolation boundary information corresponding to the complete TIFID and resumes the flow rather than create a new one. The exchange of SYN and SYN+ACK packets in this case allows the stacks to re-synchronize by exchanging the TISeq numbers where they left off at the time of the disconnection. Re-synchronization attempts are also validated by TISeq numbers that logically fit within the current state of the flow similar to how TCP validates sequence numbers.

[3]Note that unlike conventional TCP options, the IBOs are acknowledged in the above exchange as part of the three-way handshake. If the options in either direction were removed, by a middlebox perhaps, then both stacks would become aware of it and fall back to legacy behavior.

3.1.4 Use of the Control Channel

At the conclusion of a successful setup phase, a control channel exists between the two stacks. At this point, data channels to serve the application's data stream will be set up subordinate to the control channel. These are set up in a similar fashion to the control channel and are separate transport-independent flows. Requests and responses that implement additional functionality on top of TCP are communicated across the control channel. The following are some of the possiblities. In each of the cases, the control channel provides a mechanism for composing or manipulating transports of various kinds.

Resuming After a Disconnection. The ability to resume after a disconnection is a direct consequence of implementing the Isolation Boundary and is discussed in Section 3.1.3.

Multihoming. Such support can only be possible if we decouple flow identification from the transport endpoint identification. Since we can do so given the Isolation Boundary, we can construct a virtual flow which may be mapped to transport connection over different networks. This requirement was also identified in MPTCP [2].

Sophisticated possibilities of striping a virtual flow onto multiple transport connections (operating over different network paths) may also be realized — for reliability purposes. In these cases, the question of how to map the transport independent sequence space onto multiple TCP sequence spaces must also be carefully considered.

Migration. Given the extensions of resuming after a disconnection and support for multihoming, we can envision the possibility of migrating the mapping of a virtual flow from one transport connection to another.

Hybrid Transports. With a control channel in place, the communicating peers can construct a hybrid transport. The control channel allows the peers to share information regarding appropriate application gateways. While the peers converse over a (typical) packet-switched transport connection, the respective gateways may be requested (by the peers) to setup a circuit on their behalf. Once the circuit is in place, the peers may

setup transport connections to the application gateways and later migrate the virtual transport for the packet-switched transport to the hybrid transport connection.

The precise protocol for the control channel is the subject for future discussion among the community but it is envisioned that the protocol will be extensible in order to remain flexible in the face of future requirements. Clearly, setup and teardown messages will be required. The means for determining the capability of the remote peer will also be necessary.

3.1.5 Lightweight Isolation Boundary Operation

Up to this point, the isolation boundary option defined for TCP provides increased functionality by creating a control channel that is used by the stacks to negotiate and implement new functionality. Not all transport connections need the full extensibility (and heavier weight) of a control channel. We therefore define a variant of the IBO which still provides transport independence for data but does not create an out-of-band control channel. To distinguish between the two variants, we call the first *Isolation Boundary Option – Control (IBO-C)*, the variant discussed so far, and the second *Isolation Boundary Option – Data (IBO-D)*. Both types are established in the same way as discussed in Section 3.1.1. IBO-C allows protocol designers great flexibility in adapting TCP's behavior. Normally IBO-D is used for a subordinate data channel, but in lightweight operation has no associated control channel. Two stacks negotiate the use of light-weight operation if either stack advertises an IBO-D during the initial flow setup. This admits a simpler transitional implementation.

The IBO-D establishes an opaque flow onto which an application data stream is mapped. Because the flow is opaque, the only capability added to TCP by the option is the separation of the transport-independent flow from the underlying transport connection. However, this is sufficient for IBO-D to support resuming after disconnection and migration. These capabilities are useful to applications even if there is no need for any other functionality.

3.1.6 Mapping Sequence Spaces

A one-to-one mapping of the transport-independent to the TCP sequence space is straightforward. Connection setup establishes the initial mapping. During a transfer, the sequence numbers advance in synchrony as data is successfully acknowledged by the transport layer. Because of the implicit synchronization, there is no need to explicitly send the TISeq and TIAck numbers after the three-way handshake.

The synchronization between the TISeq and corresponding TCP sequence numbers is lost if the transport connection is lost. During reconnection, the correspondence between the TISeqs and the new TCP sequence numbers is re-established, thereby resuming reliable communications at the same point where the flows left off in the application data streams.

We note that other mappings between the transport independent and TCP's sequence numbers are possible depending upon the functionalities being implemented by the stack over the control channel. For example, multiple data connections may be setup and data striped across the the connections to take advantage of multiple paths for resilience or throughput purposes.

3.2 A Strawman Wire Protocol

To assess the feasibility of a backward-compatible TCP isolation boundary, we now imagine one possible implementation of the TCP isolation boundary option.

We define an option with explicit fields for the TIFID, TISeq, and TIAck. The full compliment of fields are likely not needed in each phase of connection establishment. As such, the bit field definitions below can be considered a worse case, consuming most of the remaining option space during connection setup. A more frugal mapping of concept to bits is certainly possible.

A TCP header may consist of up to 40 octets of options. Over the years, a number of options have been defined. Hence, the space available for new options has become constrained. Note that even though almost the entire remaining options space during establishment is consumed for the IBO, the role of TCP options to allow for extensibility can now be assumed by the control channel in a far more flexible manner.

At connection setup time, there are already four TCP options in common use: window scaling, time stamps, maximum segment size, and selective acknowledgments permitted. Factoring in the 19 octets these options require, 21 octets out of 40 are still available during connection establishment. A simple approach uses 20 of the remaining octets to implement the isolation boundary.

The first field of 48-bits contains the *Transport-Independent Flow Identifier (TIFID)* which labels the flow independent of the underlying transport. As the TIFID only needs to be unique within the context of the two end hosts, the requesting process specifies a locally unique value for the first half of the TIFID and the responding process later specifies the second half of the TIFID. Thus the TIFID is guaranteed to be unique to both stacks. During the time that the TIFID is partially specified, the second half is set to zero.

The isolation boundary between the upper protocol layers and the transport is further strengthened by two 48-bit protocol-independent sequence spaces, one for each flow direction.[4] As with TCP, the two endpoints select initial *Transport-Independent Sequence Numbers (TISeq)* during the three-way handshake. *Transport-Independent Acknowledgement Numbers (TIAck)* are returned to acknowledge the receipt of the SYN packets.

The TISeq are mapped onto the protocol-dependent sequence numbers of the underlying (TCP) transport and remain synchronized with them as long as the transport connection is active. When a transport sequence number is incremented, so is the TISeq.

4. CRITICAL ANALYSIS

Any change to TCP will appropriately be met with a critical eye. Protocols and practices have evolved from the widespread use and development of the Internet creating dependencies on TCP. We have argued that TCP should admit a smooth transition to new functionality in order to minimize the cost of transition for network operators and users. In this sense we now turn our attention to the interaction with existing protocols and practices. We assume for the

[4]The TIFID and the sequence number fields were chosen to be as large as possible as a compromise between providing better support for large congestion windows and the number of option bits available.

sake of this analysis the strawman wire protocol previously described.

4.1 TCP Option Space

In considering the plausibility of an IBO for TCP, it was critical to assess the space available in the TCP header. We focus on options that are considered mandatory or in common use in implementations. As a distinction, the validity of many options depends on the state of the connection. During the three-way handshake the following options need to be supported: Maximum Segment Size (RFC793, four octets), Window Scaling (RFC1323, three octets), Selective Acknowledgement Permitted (RFC2018, two octets), and Time Stamp (RFC1323, ten octets). Based on our analysis, there is sufficient room for the 20 octets that the two isolation boundary options require.

4.2 Incompatible Options

Due to the limited TCP option space, not all options can be supported simultaneously. Here we address several other options that are valid during the three-way handshake. We will disregard the Alternate Checksum Option (RFC1146) and the Partial Ordering Option (RFC1693) since according to the TCP Roadmap (RFC4614), there is a lack of interest in these protocols. The TCP Roadmap also notes that T/TCP (RFC1644) has a serious defect. TCP MD5 (RFC2385) and the follow-on TCP Authentication Option (RFC5925) are used to protect BGP and LDP and hence are likely not to benefit from an isolation layer. Since these are concerned with protecting the infrastructure itself and are not used for user traffic, we need not concern ourselves with compatibility. The last protocol we consider is the Quick-Start Response (RFC4782). This protocol is experimental, and it remains to be seen if there will be widespread adoption. If the IBO and Quick-Start were both to come into common usage, then the contention will need to be resolved by omitting some option from the SYN and SYN/ACK packets.

4.3 Performance

The lack of field alignment, regardless of which option causes it, may lead to degraded performance for some network stacks due to the misaligned memory accesses that may require individual octet manipulations. In the case of the IBO, a peer stack may see degraded performance whether or not it supports isolation. Since the IBO is only valid during the three-way handshake, their processing is off the critical data path and thus should not adversely affect performance.

4.4 Simplicity

With regard to simplicity, the isolation boundary directly implements only (1) the decoupling of a transport independent flow from the transport connection, and hence, from the network endpoint identifiers involved, (2) provides a means for keeping track of where the conversation is in the flow, and (3) provides for the establishment of a control channel upon which additional functionality can be built. The immediate implication is that multiple TCP connections can be associated with one flow over its lifetime with multiplexing opportunities in both time and space. Clearly the first two make re-synchronization possible. We go no further with the third at this time other than to acknowledge that the possibility of using the control channel to extend the service TCP offers through inexpensive, out of band signaling. This is done with the expectation that other protocol designers will use this mechanism to extend TCP's functionality in the future.

4.5 SYN Cookies

SYN cookies [21] mitigate a serious vulnerability in TCP. A server must maintain state in its SYN cache for each connection attempt received. An attacker can easily exploit this and fill up the SYN cache by simply crafting TCP SYN packets to overwhelm the server. Normally the SYN cache records the state that is required to establish a TCP connection. The IBO would also have to be recorded in the SYN cache. When a server is under attack, it instead responds with a SYN cookie that maintains minimal state and allows the server to continue to serve new connection requests but in a degraded mode of operation. Like most options, the IBO will not be preserved when a server is operating under attack. Because the isolation boundary is backward compatible, a server in this mode will continue to operate in a classic TCP fashion.

4.6 Middleboxes

A TCP connection that makes use of the IBO behaves identically to a TCP connection that does not. The primary concern with respect to the IBO is intrusion detection/prevention systems. If one of these systems encounters an as yet unknown IBO, it may do one of two things. It may either remove the offending option or it may discard the offending packet. From the concern of not degrading responsiveness as perceived by the user, the first option is preferred. The first option will simply cause TCP to fall back into a classic operation mode. The second, however, will be a needlessly dropped packet that forces the user to wait for retransmission in order to fallback to a legacy mode of operation. This would be perceived by the user as unresponsive but could be mitigated by the client sending two SYN packets, one with and one without the IBO. The client would then prefer the connection that supported Isolation and simply reset the other.

4.7 Security

As a design goal, the isolation boundary should have security characteristics that are no worse than TCP. The primary vulnerability of use of the IBO is that it allows resynchronization of a connection from any network address. If an attacker knows critical information about the current connection state, it is possible to hijack an existing connection from anywhere else on the Internet, but an attacker must know the TIFID and the TISeq numbers of the current set of unacknowledged data.

In order for an attack to be successful, the attacker must have knowledge of the full conversation from the point of instantiation. The TIFID is only exposed in the three-way handshake and cannot be easily guessed due to its length. The TISeq numbers need to be derived from the initial sequence numbers, the current TCP sequence numbers, and the fact that TCP sequence number roll-over may have occurred. In other words, the attacker needs to know the TIFID and the count of all the octets that pass by in either direction in order to falsify the reconnect request.

If the attacker is not on the data path between the parties in the connection, then the attacker must have a collaborator

that is. This is no worse than TCP connection hijacking without the IBO in that the attacker in this case also needs to be in the data path. TCP hijacking is more onerous in this instance because the attacker need not be privy to the beginning of the connection.

4.8 Application Compatibility

Not only does an extended TCP that utilizes an isolation boundary need to be backward-compatible with other peer implementations, but it also needs to be backward-compatible with applications. Since in all other respects the semantics of TCP have not changed, an application that is unaware of other functionality that might be enabled by the isolation boundary will continue to operate correctly when using a TCP with the IBO. In fact, this application will gain some benefit in being able to re-synchronize a lost connection.

5. SUMMARY AND FUTURE WORK

In this paper, we have laid out an argument for establishing an Isolation Boundary for TCP that maintains backwards compatibility. Clearly much work is still required to fully evaluate the approach and we have only just begun work on an implementation.

The specification of the control channel protocol has been intentionally left out of scope for this work. However we feel that a compliant stack that implements the Isolation Boundary must admit the possibility of a control channel and properly negotiate a data only channel in addition to implementing the control channel itself. We have claimed that given an Isolation Boundary, protocol designers will be able to construct higher level functionality on top of TCP. As a proof of this claim, we will at least need to create a mock implementation of the control channel and construct some higher level functionality on top of this.

A central goal of this protocol is backward compatibility. This in and of itself needs to be surveyed against existing implementations and across existing networks. Finally in the spirit of the IETF, multiple independent implementations need to be constructed and put to a large scale test of interoperability.

Acknowledgements

This research was funded in part by Juniper Networks. We thank Colin Constable, Jayabharat Boddu, Danny Jump, and Barnaby Crahan for their feedback and support. We also thank the reviewers for their helpful suggestions.

6. REFERENCES

[1] V. Cerf, "Internet Predictions: Future Imperfect," in *IEEE Internet Computing*. IEEE, 2010, pp. 30–33.

[2] A. Ford, C. Raiciu, M. Handley, and S. Barre, "TCP Extensions for Multipath Operation with Multiple Addresses," RFC (Experimental), Work in Progress, Internet Engineering Task Force, Jul. 2010.

[3] J. Iyengar and B. Ford, "A Next Generation Transport Services Architecture," RFC (Informational), Work in Progress, Internet Engineering Task Force, Jul. 2009.

[4] B. Ford and J. Iyengar, "Efficient Cross-Layer Negotiation," in *HotNets-VIII*, 2009.

[5] E. Kissel, M. Swany, and A. Brown, "Improving GridFTP Performance using the Phoebus Session Layer," in *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. ACM, 2009, pp. 1–10.

[6] B. Ford and J. Iyengar, "Breaking up the Transport Logjam," in *HOTNETS-VII*, 2008.

[7] A. Brown, M. Swany, E. Kissel, and G. Almes, "Phoebus: A Session Protocol for Dynamic and Heterogeneous Networks," University of Delaware, Tech. Rep. 2008:334, 2008.

[8] B. Ford, "Structured Streams: A New Transport Abstraction," in *ACM SIGCOMM CCR*, vol. 37, no. 4. ACM, 2007, pp. 361–372.

[9] A. Habib, N. Christin, and J. Chuang, "Taking Advantage of Multihoming with Session Layer Striping," in *IEEE INFOCOM*. IEEE, 2006, pp. 1–6.

[10] T. Mahieu, P. Verbaeten, and W. Joosen, "A Session Layer Concept for Overlay Networks," in *Wireless Personal Communications*, vol. 35, 2005, pp. 111–121.

[11] J. Salz, A. C. Snoeren, and H. Balakrishnan, "TESLA: A Transparent, Extensible Session-Layer Architecture for End-to-end Network Services," in *4th USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003, pp. 211–224.

[12] A. C. Snoeren, "A Session-Based Architecture for Internet Mobility," Massachusetts Institute of Technology, Tech. Rep., 2003.

[13] A. Snoeren, H. Balakrishnan, and M. Kaashoek, "Reconsidering Internet Mobility," in *Hot Topics in Operating Systems*, 2001, pp. 41–46.

[14] B. Landfeldt, T. Larsson, Y. Ismailov, and A. Seneviratne, "SLM, A Framework for Session Layer Mobility Management," in *8th International Conference on Computer Communications and Networks*, 1999, pp. 452–456.

[15] D. A. Maltz and P. Bhagwat, "MSOCKS: An Architecture for Transport Layer Mobility," in *IEEE INFOCOM*, vol. 3. IEEE, 1998, pp. 1037–1045.

[16] K.-K. Yap, M. Kobayashi, D. Underhill, S. Seetharaman, P. Kazemian, and N. McKeown, "The Stanford OpenRoads Deployment," in *4th ACM WINTECH*. ACM, 2009, pp. 59–66.

[17] F. Teraoka, "Redesigning Layered Network Architecture for Next Generation Networks," in *IEEE GLOBECOM Workshops*. IEEE, 2009, pp. 1–6.

[18] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers *et al.*, "A Clean Slate 4D Approach to Network Control and Management," in *ACM SIGCOMM CCR*, vol. 35, no. 5. ACM, 2005, pp. 41–54.

[19] A. Greenberg, "Refactoring Network Control and Management: A Case for the 4D Architecture," Carnegie Mellon University, Tech. Rep. CMU-CS-05-117, 2005.

[20] A. C. Snoeren and H. Balakrishnan, "An End-to-End Approach to Host Mobility," in *Proceedings of the 6th Annual International Conference on Mobile computing and networking*, ser. MobiCom '00. ACM, 2000, pp. 155–166.

[21] D. J. Bernstein, "SYN cookies," February 2002. [Online]. Available: http://cr.yp.to/syncookies.html