Masking Network Heterogeneity with Shims

Danny Yuxing Huang*, Eric Kimbrel®, Justin Cappos*, Jeannie Albrecht*
* Williams College, Williamstown, MA   ® University of Washington, Seattle, WA
{yh1, jeannie}@cs.williams.edu, {kimbrl, justinc}@cs.washington.edu

The Internet has evolved from the initial four-node set-up to the diverse and unpredictable environment that exists today. This provides programmers with the challenging task of building applications that work in heterogeneous network environments that include NATs, VPNs, firewalls, mobile devices, and open network connections. The state-of-the-art solution to handling network heterogeneity is to port a program to use a library that handles a specific type of network heterogeneity. For example, suppose an application needs to support incoming connections even when the system is behind a NAT. The programmer may change the networking code in the application to use STUN or similar to bypass this issue. Similarly, to support mobile computers, the programmer may modify the program in order to interface with the Java Mobile Socket, or Mocket, that handles unexpected changes in the IP address. Porting an application to support the interfaces provided by these libraries can be time consuming and error prone because the libraries do not follow the semantics of the underlying system network API. In other words, while both tend to share a similar interface (to reduce the difficulty of porting code), the exact behaviors of individual methods can potentially be different. Furthermore, a major challenge occurs if we need to use both libraries concurrently. A mobile application using the Java Mocke can be running behind a wireless NAT, and we cannot simply layer STUN on top of Mocket, because the two APIs do not agree on semantics (Figure 1). In addition to modifying the application to use the library, we must also modify one library to use the interface provided by the other library to compose their functionality. Hence, the lack of a consistent set of semantics in generic libraries poses a challenge to application developers.

In our work, we abstract away network heterogeneity by encapsulating network functionality in a formally-verified wrapper called a shim. A shim provides functionality like NAT traversal or mobility support through a semantically identical interface to the underlying network API. As a result, the shim is completely transparent to the application and no modification to the original code is necessary. For instance, we have constructed a NAT traversal shim that enables a server application to accept incoming connections even behind a NAT device. The shims on the server and the client actually make the connection via a proxy in the public network. To the application, however, the proxy is invisible and the communication appears as if it were directly established. The application behaves correctly regardless of how the shim is actually implemented, because the shim follows the semantics of the underlying network API. Since shims provide and use interfaces with the same semantics, one may compose the functionality of multiple shims on the same application without modification to either the application or shims (Figure 2). This allows the application developer to focus on the functionality of the program, while shims address network heterogeneity.

Currently, we have an preliminary deployment of shims on the Seattle Testbed. Based on our preliminary deployment experience, making an application shim-capable is easy. For instance, any program written for the Seattle Testbed can be programatically rewritten to use shims without any manual intervention. With shims, applications are able to operate in more diverse environments and achieve better performance. We have built a type of shims called deciders that can dynamically push the appropriate shims based on network and system resources. The NAT decider shim, for example, pushes the NAT traversal shim only when a NAT is detected. This reduces the unnecessary latency that would otherwise occur as a result of connecting through the proxy. We have also constructed a number of shims, including: the logger shim, which records the sources and destinations of all TCP traffic; the encryption shim, which encodes/decodes all server-client communications with a public-private key-pair; the compression shim, which compresses data and thus allows greater throughput; and the reverse connection shim, which enables a NAT’ed server to listen for incoming connections by actually making direct outgoing connections to the clients. We plan to deploy the shim-enabled part of the testbed to the DieselNet buses, where our shims’ capability of abstracting away mobility will be put to test. We would like to perform a live demo of our system at the poster session, where the audience can observe, in real time, how the same application can adapt to various network environments, while the shims silently handle all these changes.

1 Undergraduate student at Williams College.
2 http://seattle.cs.washington.edu
3 http://prims.cs.umass.edu/dome/umassdieselnet