Decoupling Port and Protocol on the Internet

Kyle King, Michael Lam, Aaron Schulman
{kking,lam,schulman}@cs.umd.edu

December 14, 2007

Abstract

Well-known ports are the current convention for protocol-to-port association in transport layers today. This convention complicates service discovery and limits development by requiring registration of new protocols with a centralized authority. Ultimately, developers have subverted this convention by layering traffic over HTTP.

We propose a wrapper for the socket API that provides a string-based application naming scheme, and removes the need for network applications to be aware of port number. This will increase separation between the transport and application layer, alleviating the previously mentioned difficulties while opening up new areas of stakeholder interaction on the Internet.

1 Introduction

The Internet relies on the coupling of networked application and transport protocol port in order to multiplex traffic above the network layer. The original transport control RFC specified that connections would be identified by a socket, which is a tuple composed of host ID, TCP identifier, and "port identifier" [1]. The authors originally used numbers as port identifiers because it was a simple solution and was sufficient for the small number of applications at the time [13]. This concept of well-known ports (WKPs) survives in the current Internet. However, development of new Internet applications has proved to be more dynamic than the authors of TCP envisioned.

Firewalls and NATs identify, block, and forward traffic based on the transport-layer port number rather than an application-layer identifier. Networked applications like Skype [14] have methods to bypass these port-based security measures.

Portmap [15] and DNS SRV [6] based systems like Bonjour [3] are partial solutions that provide service port discovery in the absence of assigned port numbers. However, these solutions still require application developers to bind their application to a single, consistent port.

As a solution, we propose (1) a daemon responsible for service discovery and (2) a shared software library that performs service registration and discovery tasks. The library will allow applications to dynamically register with the daemon service, enabling service discovery using a string-based naming scheme. The applications will then use the shared library to abstract away socket programming. The library will handle all connection management and hide the actual port number used by the underlying TCP/UDP connections. In this way, port numbers are used strictly as session identifiers.

Our solution will open a new "tussle space" [4] on the Internet between application developers and network security teams, and will create new opportunities in privacy, simplified application development, and namespaces.

The structure of the remainder of the paper is as follows: we will first discuss the problem and its origins as well as related work as a starting point for the development of our own solution. We will then explain the details of our design and how we implemented it. Finally, we will make the argument that the overall design represents a significant improvement over the current state of the art.

2 History

To understand the problem that our work addresses, it is useful to examine the origins of TCP and well-known ports (WKP).

The authors of TCP originally presented the idea of well-known sockets to provide basic service discovery [1]. At the time of TCP's creation, the authors were focused more on building a reliable transport protocol than on creating a service discovery and application-layer protocol identification system [13]. The WKP solution was easy to implement, and it was reasonable for the Internet at the time since there were only a couple of processes running on each host.

This "expedient convention" [13] was never intended to be permanent. By 1972, the convention was already in widespread use and RFC 322 [2] established Postel and Cerf as the first official maintainers of an ad-hoc well-known ports list. Postel published the list and developers could ask to have their applications added. In 1990, Postel and Reynolds published the list of allocated numbers as an RFC, calling it the "Internet Assigned Numbers Authority" (IANA) [12]. Later, ICANN took over the administration of this list.

3 Problem

3.1 Well-Known Ports Offer Limited Service Discovery Capabilities

Well-known ports (WKPs) have limitations that prevent them from being an ideal solution to the service discovery problem.

First, WKPs limit a host to running one instance of an application. For example, port 80 is the well-known port for the World Wide Web, and two instances of a web server cannot run on a single host since there is no way to distinguish between the them. Similarly, there is no way to distinguish between different versions of a service.

Secondly, there is no method to distinguish between a protocol and the applications that use that protocol. For example, voice over IP (VoIP) uses the same session initiation protocol (SIP) as some instant-messaging applications. Because all SIP traffic is bound to a single port, users typically cannot run both applications at once.

Finally, a numeric port offers no semantic interpretation a priori. Numeric identifiers are difficult to remember and cumbersome to use for ad-hoc service identification.

3.2 Centralized Management of Service Identifiers is Ineffective

There are several problems with using a central management point (IANA) to coordinate service discovery.

First, centralization suffers from the problem of stagnation. The current well-known ports list contains many protocols that are never used and/or owned by organizations that no longer exist [7].

Secondly, the amount of work required to obtain a well-known port is unreasonable given the dynamic nature of modern Internet application development, and the benefits are too ill-defined to be a compelling motivation. The registration process of a bureaucratic central authority stifles innovation by putting an excessive burden on programmers during the development of new applications.

Further, IANA offers no supporting infrastructure for service discovery. There are other centralized services that depend on a single point of registration. DNS is an example of such a service that has been widely successful. The difference between DNS and WKP is that the DNS service is dynamic and supported by a worldwide infrastructure with multiple levels of resolution. WKP lists are static and there is no supporting infrastructure aside from distribution of the list with with operating systems.

All of these problems prevent IANA from being an effective means of port-to-protocol association.

4 Solution

This paper presents Moniker, a solution that decouples port and protocol by randomly assigning port numbers to services at run-time.

Using Moniker, network applications register and request services using a semantically-meaningful string to explicitly identify applications and their versions. In in the new model, registration of service identifiers is now the responsibility of the endpoints, rather than a central management point (such as IANA).

4.1 Key Elements

Our solution is characterized by two main attributes: (1) we use string-based service identifiers instead of numbers, and (2) we assign random unused port numbers to services.

Our goal was to destroy all port/protocol association and invalidate any assumptions a developer might make concerning the ports their application is using. Moniker uses port numbers purely for session identification, and the system itself will manage all service-level identification and discovery.

4.2 Design

Our system contains two pieces: (1) a background daemon and (2) a shared library.

The background daemon runs on a host and maintains a list of all services offered by applications on that host. For backward compatibility, the daemon runs on a designated port (ex. port 1) and listens for incoming service requests. When the daemon receives a service request, it looks up the service string in its list of registered applications and sends the port number of the desired server application to the client.

The shared library abstracts away port numbers from application development by brokering sessions and abstracting all socket system calls. Server applications use the shared library functions to register a service with the daemon. The application specifies a service identifier string and the library selects a random unused port number. The library then registers this information with the daemon, which stores the identifier-to-port association.

Clients can make service requests using the Moniker library by specifying a host name and service identifier. The library then sends a service query message to the server daemon on the remote host and the daemon responds by providing the port number of the desired application. Both sides then create sockets to establish a session. After the session has been created, the library provides wrappers for standard socket functionality. Since applications communicate using Moniker-wrapped socket calls, they are never aware of the port number being used. This achieves our goal of port and protocol decoupling.

4.3 Implementation

Table 1 Daemon comparison

	portmap	Bonjour	monikerd
Service-port association	X	X	X
Host-specific query for services	X		X
Dynamic application registration		X	X

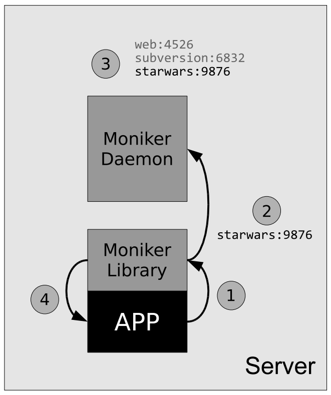
We considered using two existing daemon services (Portmap and Bonjour) to perform service/port association, but both of them had significant deficiencies stemming from their underlying libraries. In the end we decided to implement our own service to provide all the needed functionality, and here we present a comparison of the three services (see Table 1).

Portmap was designed to be used for remote procedure calls, and it provides a simple mechanism for service discovery by associating service names with their corresponding port numbers. However, Portmap lacks the ability to dynamically register new services, since all service associations are initialized statically using configuration files.

Bonjour is Apple's zero-configuration service discovery framework. It provides a multicast-based framework for host and service discovery. Although Bonjour possesses more functionality than Portmap, it lacks the ability to send port queries directly to a specific host. This feature is crucial to our system, but Bonjour requires a centralized network server to broker this queries.

The final implementation of the Moniker daemon combines the functionalities of Portmap and Bonjour to provide the three crucial features needed for our system to work: (1) service-to-port association, (2) dynamic service registration and management, and (3) direct host-to-host querying.

Figure 1 Server Registration Process

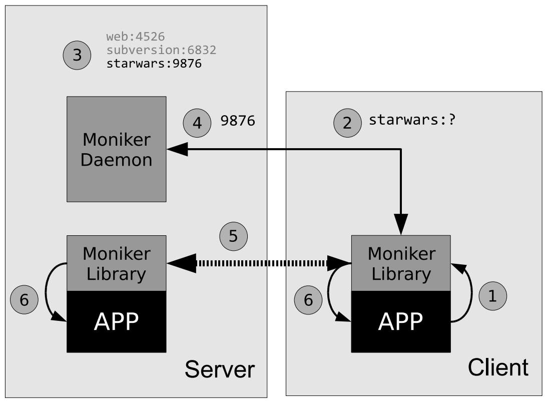


Listings 1 and 2 show a sample implementation of a moniker based application.

The process for registering a service called "starwars" with Moniker is as follows (see Figure 1):

1. The server application calls mon_service("starwars")
2. The library binds a new socket to a random port (assume it selects port 9876) and notifies the daemon of the new service
3. The daemon stores the association between port 9876 and "starwars"
4. The library returns a service handle to the server application

Figure 2 Client/Server Connection Process



The process for establishing a connection between a client and the "starwars" server is as follows (see Figure 2):

1. The client application calls mon_connect(hostname, "starwars")
2. The Moniker library sends a query for the "starwars" service port number to the remote Moniker daemon
3. The server daemon looks up the port number for "starwars" in the association table
4. The server daemon sends the port number back to the client's Moniker library
5. The client's Moniker library makes a connection to the server's Moniker library using the given port number
6. On both sides, the Moniker library returns a handle for the new connection

We believe that this prototype realizes the original service-discovery vision of the authors of TCP, since the authors intended network communication to resemble inter-process communication:

Processes are viewed as the active elements of all HOST computers in a network. Even terminals and files or other I/O media are viewed as communicating through the use of processes. Thus, all network communication is viewed as inter-process communication [1].

Listing 1 Sample Client

```
    ...
    moniker server;
    struct sw_request req;
    struct sw_response resp;
    int ret;
    ...
    ret = mon_connect(&server, argv[1],
            SERVICE_STRING, MON_STREAM,
            SERVICE_VERSION);
    ret = mon_send(&server,
            (char*)(&req), sizeof(req));
    ret = mon_recv(&server,
            (char*)(&resp), sizeof(resp));
    ret = mon_close(&server);
    ...
```

5 Implications

5.1 Security

Firewalls typically implement basic security by filtering packets based on their source and destination port. After the deployment of Moniker, this port-based identification of application protocols will no longer be possible. This fundamental shift in communication processes demands a fresh re-evaluation of network security methods.

There are three ways to secure a network in which end hosts connect to services using Moniker. First, Moniker-aware packet filtering software can attempt to identify packets in the middle of the network by examining the Moniker query and connection packets. This type of filtering would have the same pitfalls as port-based filtering today. Another method

Listing 2 Sample Server

```
    ...
    moniker server, client;
    struct sw_request req;
    struct sw_response resp;
    int ret;
    ...
    ret = mon_service(&server, MON_STREAM,
            SERVICE_VERSION, SERVICE_STRING);
    ret = mon_listen(&server, 100);
    while (1) {
        ret = mon_accept(&server, &client);
        if (fork() == 0) {

            ret = mon_recv(&client,
                    (char*)(&req), sizeof(req));

            ...

            ret = mon_send(&client,
                    (char*)(&resp), sizeof(resp));

            ret = mon_close(&client);

            return EXIT_SUCCESS;
        }
    }
    ret = mon_close(&server);
    ...
}
```

is to filter based on deep packet inspection, which attempts to identify traffic using various content-based techniques [9, 8]. A final option is to ignore security issues in the middle of the network, and focus instead on securing end hosts.

The only viable options will be either to improve deep packet inspection firewalls or properly secure end hosts. A Moniker-aware firewall can only identify the service of a flow if the initial session discovery exchange is sent in clear-text and the service string accurately identifies the service being used, while deep packet inspection firewalls can filter traffic without observing any service discovery exchanges. Securing the end hosts provides the highest level of protection in a Moniker-based system. Moniker policy can be set to limit end hosts to connections with authorized services. Also, there have been an increasing number of automatic update systems built into operating systems and other software in order to keep end system software up-to-date and secure.

It is probable that malicious users will attempt to bypass service-based security by tunneling over accepted services, but this is not a problem unique to Moniker since port 80 tunneling is widely considered the status quo today. Our proposal simply formalizes and clarifies the security situation, acknowledging that deep packet analysis has always been the only way of truly identifying network traffic, and that securing end hosts will prevent them from being compromised no matter how inadequate the middle-of-the-network security is.

Widespread use of Moniker could prompt a rebirth in the network security community by forcing the development of better content-based analysis methods and more effective ways of securing end hosts.

5.2 Application Development

Our proposal has four significant application development implications.

First, our proposal eases the actual process of network programming. Instead of being forced to use a port number that lacks semantics, the application developers can choose a string-based identifier that more uniquely and intuitively represents their application. Furthermore, the Moniker library provides a wrapper for many socket functions, which reduces the amount of code needed to create a networked application.

Second, Moniker enables developers to quickly swap out transport protocols. Previously, developers would have needed to tunnel traffic over the existing protocol (as with TCP and SNA [11]) or add the new protocol functionality to all client programs (as with SCTP [5]). With Moniker, the protocol developer would only have to add their code to the shared library and the functionality would automatically be available to all applications.

Third, URIs are simpler to remember and more descriptive. The intuition and simplicity of URIs have made them increasingly common in networked applications. Moniker can take advantage of URI's, making service discovery even simpler. For instance, instead of identifying the "starwars" server on port 9876 using "theforce.net:9876", a user would specify the URI "starwars://theforce.net". The latter is easier to remember because it explicitly names the service requested.

Fourth, the general deployment of applications is far easier once port and protocol are decoupled. This is partially because programmers no longer have to send all traffic to and from a single allowed port (ex. port 80) [10] or apply for their IANA-registered port to be opened. In addition, there is no longer a central registration point that must approve a port assignment. Each application self-advertises to the host.

5.3 Namespaces

Currently, URIs identify shared resources on a network. For instance, a user would make a request for the IETF website using the following command:

```
lynx http://www.ietf.org
```

This command requests HTTP from the IETF server. Applications actually interpret the the "http://" portion of the string as meaning "port 80," using some built-in association between "http" and "port 80."

Moniker uses the same URI syntax, but adds operation semantics. Using Moniker, the URI library can simply query the remote host to obtain a service's port number instead of consulting a local port association table. For the previous example, the client Moniker library will query the "www.ietf.org" host for the port number associated with the "http" service, and will then create a connection with the HTTP server.

This new URI method easily lends itself to the addition of namespaces. For example, if one wanted to access Joyce Reynolds' web server on the IETF domain today, the current solution would require a subdomain or HTTP redirect. With an extension to Moniker for parsing namespaces, the background daemon on the server would be able to interpret the following request:

```
lynx reynolds::http://ietf.org
```

This allows an arbitrary number of web servers (or any other kind of service) to run on a single domain, using namespaces to distinguish between them. Namespaces can also be used to identify and forward service requests to users behind NATs, similar to Apple's PMP-NAT protocol.

6 Future Work

Currently, our prototype supports the TCP and UDP transport protocols. We would like to add support for the SCTP protocol [5] to show that using Moniker, developers can create applications that are agnostic to the transport layer. This would allow existing applications to utilize new transport protocols with a minimal amount of modification. We are also interested in integrating our system into URI resolution libraries so it could be used by current network applications.

Currently, all Moniker service requests are unencrypted. To increase privacy, we would like to add the option of encrypting service requests. This would make it very difficult to identify an encrypted network flow since there would be no clear-text session initiation packet.

Finally, as a larger goal, we envision the addition of a distributed, self-seeded, endpoint name resolution service similar to Bonjour. The combination of our system and such a framework would move all naming responsibilities to peers rather than a central registry. With this sort of system in place, a user could query entirely by content, rather than by destination. The name resolution service would oversee the routing of the packet to the proper host, and Moniker would ensure that the connection is brokered with the proper application on that host. As peer-to-peer networks become more popular, this goal of end-to-end content-based addressing becomes both desirable and possible.

7 Conclusions

Service discovery is an important problem, and well-known ports represent a working and widely-used but non-ideal solution to the problem. We propose a more ideal solution that completely decouples port and protocol by adding a broker service that handles end-to-end service registration and session management. By randomly assigning port numbers, we invalidate any assumptions a developer may make about port-to-protocol associations.

We believe the adoption of a system such as Moniker will have several benefits: (1) network application development will be simpler, (2) network administrators will need to re-think security, and the "tussle" between administrators and malicious users will be clearer, and (3) namespaces will provide a new way to identify service instances.

Acknowledgments

The authors would like to acknowledge the valuable suggestions and feedback of Justin McCann.

References

[1] V. Cerf, Y. Dalal, and C. Sunshine. Rfc 675: Specification of internet transmission control program.

[2] V. Cerf and J. Postel. Rfc 322: Well known socket numbers.

[3] S. Cheshire, B. Aboba, and E. Guttman. Dynamic configuration of ipv4 link-local addresses.

[4] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden. Tussle in cyberspace: defining tomorrow's internet. *IEEE/ACM Trans. Netw.*, 13(3):462–475, 2005.

[5] R. S. et al. Rfc 2960: Stream control transmission protocol.

[6] A. Gulbrandsen, P. Vixie, and L. Esibov. A dns rr for specifying the location of services (dns srv).

[7] E. Lear. Port assignment procedures (ietf draft), 2006.

[8] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M. Voelker. Unexpected means of protocol inference. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 313–326, New York, NY, USA, 2006. ACM.

[9] A. W. Moore and K. Papagiannaki. Toward the accurate identification of network applications. In *Passive and Active Network Measurement 2005 (Lecture Notes in Computer Science series)*, volume 3431, pages 41–54. Springer Berlin/Heidelberg, 2005.

[10] K. Moore. Rfc 3205: On the use of http as a substrate.

[11] D. Ogle, K. Tracey, R. Floyd, and G. Bollella. Dynamically selecting protocols for socket applications. *Network, IEEE*, 7(3):48–57, May 1993.

[12] J. Reynolds and J. Postel. Assigned numbers.

[13] G. Skinner. What if there were no well known numbers? Postel.org forum posts, August 2006. `http://www.postel.org/pipermail/end2end-interest/2006-August/006123.html`.

[14] Skype limited skype and firewalls. `http://www.skype.com/help/guides/firewalls/technical.html`.

[15] R. Srinivasan. Rfc 1833: Binding protocols for onc rpc version 2.