fived: A Session Layer Design to Secure the Internet

Technical Report UCSC-WASP-15-01
November 2015

D J Capelis
mail@capelis.dj

Working-group on Applied Security and Privacy
Storage Systems Research Center
University of California, Santa Cruz
Santa Cruz, CA 95064
`http://wasp.soe.ucsc.edu/`

Abstract

This technical report outlines the design of *fived*. An ongoing project to bring features currently implemented in the application layer of the Internet into the core network stack. These features would traditionally describe OSI 7-layer model's session layer. This design would have the greatest impact on the security and flexibility of our networks.

1 Introduction

On today's network, an unwieldy array of different components are tasked with security responsibilities. Application developers routinely make mistakes in their security critical code, leading to bugs that manifest as worms and malware. Access control mechanisms on the network typically rely on *where* a user's computer is located, not on *who* that user is. The systems that authenticate users remain separated from the firewalls tasked with controlling access to various network services. The network is without the information required to make intelligent access control decisions. These problems are compounded by the Internet's remarkable resistance to change. Many security technologies have failed to achieve adoption over the years. *Fived* is a design for a unified session layer that integrates security features into the core of the Internet, one user, one network or one application at a time.

Let's begin with the problem of access control. Firewalls, the main source of access control in most deployed networks, dictate access control policies based on the host's IP address. Any network that wishes to support legitimate users' ability to access services from networks not directly owned by the organization must support a mechanism to bypass the security perimeter; this is typically provided in the form of a Virtual Private Network (VPN) [12] connection. Likewise, organizations that wish to offer courtesy access to guests have to institute registration processes and network partitioning just to allow visitors to check their e-mail without exposing internal services. In more complex organizations, where users have various levels of access, clearance or affiliation, network partitioning can get even more complex, arduous and brittle.

A session layer design provides the user with the notion of a session, but more importantly, allows them to authenticate and open up the range of services available to them. This layer makes access control decisions based on who the user is, what groups or roles that user may be a part of and any number of additional policies the network administrator might wish to support. This provides the type of comprehensive access control modern networks need.

Yet the benefits of a robust session layer extend beyond simplifying the lives of network administrators and reducing the complexity of security configuration. The current Internet architecture forces each individual network application to write large amounts of sensitive code to provide security features, including authentication and encryption. Applications often simply omit some of these features, while the remainder provide a wide array of encryption and authorization solutions of varying quality in terms of usability or security. A session layer puts security features on-par with core networking concepts like congestion control. With a session layer in place, applications can take advantage of one unified codebase to perform these types of sensitive operations. Support for new authentication mechanisms, new encryption technologies, or other new security features, can be added in one place and made immediately available to all applications running on the session layer.

One of the hardest problems of changing the Internet is adoption. Worse, in the next several years, new architectures which seek to gain acceptance on the Internet face several specific challenges. IANA has allocated their last free block of IP addresses [14] to the Regional Internet Registries and each has put in place emergency procedures to manage their last remaining IPv4 addresses. Over the next years, the Internet will reel as it reactively deploys the solutions networking researchers converged on 15 years ago. Due to this unfortunate timing, any realistic deployment of new networking technologies reliant on commercial network operators to adopt new equipment, standards or practices will be challenged for several years to come. Yet the switch to IPv6 will not solve many of the pressing problems that have become more apparent over the last decade and a half since IPv6 was designed. The Internet can't wait another 15 years for new technologies to reach deployment.

In an environment where developing a compelling improvement is merely a necessary, but not sufficient condition for deployment it is critical that any proposed changes provide a realistic transition plan. *Fived*'s key deployability advantage is that it follows the end-to-end principle [22] allowing progress to trickle in from the edges of the network to the core. Potentially lethargic core network operators do little to harm the adoption of *fived*. Another key component of a deployment plan is that the system must not require a large critical mass before organizations begin seeing benefits. Users of *fived* gain some benefit right away. Downloading code is enough to allow users to start controlling access to their internal services. A set of compatibility libraries, layers and runtime tools can provide users the ability to obtain advantages from *fived* even before applications are adapted to interact with the session layer natively.

The final defining feature of a viable architecture change is its ability to lead us away from the current level of stagnation on the Internet. A new architecture change not only needs to overcome its own deployment challenges, but should attempt to open up the Internet in ways that allows increased flexibility in the future. The Internet's resistance to change isn't sustainable and without modifications, the network will not be flexible enough to head toward the future. Future technologies must be designed so if they succeed, the Internet can absorb new ideas, innovations and technologies at a higher rate than the current network.

The session layer as implemented in *fived* is extendable. Creating a new service can be as simple as picking a name for it, adding some lines to a configuration file and writing as little as 10 lines of code. This architecture is considerably more flexible than the fixed size headers most of our modern Internet protocols use today. The core services in this document should be useful for years to come, but future researchers can experiment with new variants on these services merely by adding it to their local session layer and beginning to use it. Standards can spread either organically or via vendors working together in a formal process. Innovative ideas can be demonstrated easily and adopted quickly as consensus develops.

In addition to ensuring that our own additions are flexible, *fived* aims to ensure that the underlying components of the current Internet grow easier to replace. This is the job of the session initiator, the component that establishes a session. The session initiator resolves names to addresses, deals with various transport protocols and sets up connections or an ability to send datagrams in future networks. Adding a new transport protocol or changing Internet addressing merely requires modifications to the session initiator and any application using it can adapt. This allows the layers underneath the session layer to change and accept new technologies as well.

Fived provides solutions to a broad range of recent problems, with specific focus on embedding trustworthiness into the fabric of the Internet. The design has a transition plan, a design which allows for a suite of compatibility tools and has the potential to bypass many of the roadblocks during what's likely to be a messy transition process to IPv6. Finally, the design serves as a catalyst for past, present and future technologies by ensuring that if *fived* is successful, the deployment barrier on the Internet is reduced.

2 Related Work

One of the challenging parts about explaining *fived* has been in comparing it to existing technologies. *Fived*'s design goals revolve around incorporating solutions to problems where we know the network has needs. The solutions *fived* implements often aren't particularly different than existing technologies. It isn't in the choice of encryption algorithm or the protocol that *fived*'s contributions are really understood. It is in the way this session layer enables the use of these technologies in a way that creates a coherent architecture between every application using *fived*. It is in the way that *fived* shifts the responsibility into the underlying layers and eliminates sections of security critical code required in many of today's applications. It is in the way *fived* strictly adheres to the end-to-end principles and eschews any requirement that the core networking hardware know about our protocol for it to succeed. It is in the way the session layer architecture enables application access to these technology in a uniform way across the deployment base.

Which isn't to say no other projects have had these goals. Service-oriented network designs, such as those seen in Planetlab [8] and GENI [19] often have similar design goals. Chandrashekar's paper on a Service Oriented Internet [6] comes up with a strikingly similar design in some respects. In this paper, a session layer with a service-oriented architecture is fairly clearly proposed and outlined. The main difference between these works and those of *fived* is a difference in how these systems interact with legacy technology. Many of these designs fall under the category of "clean slate" networking architectures, where the goal of the research is to clean up the Internet and switch to a "better" architecture. *Fived* on the other hand, is what I like to call a "dirty slate" design. The goal of *fived* is to add the features into the *existing* network that seem to be missing. When there's a way to do it that seems to prod the network towards a cleaner architecture, *fived* takes the opportunity, but the guiding goal is to get the features into the network. The resulting systems turn out fairly different.

On a feature by feature basis, there are many comparisons between *fived* and other systems:

Service discovery allows a computer to query whether a service is running. Traditionally this is done via attempting to establish a connection on a standardized port number and assuming that if a service exists on the machine, it will be listening there. *Fived* allows a user to instead specify services using a name, a minor improvement to usability that shifts the namespace from numbers to characters. Other software that has tried this approach includes the portmap [23] service, which protocols like NFS [24] rely on.

Encryption on the Internet is hardly a new feature, SSL [11] and later TLS [9] have been providing encryption services on the network for awhile. Newer protocol proposals, like MinimaLT [20], CurveCP [5] and QUIC [25] offer transport security with improved cryptographic properties. *Fived* implementations can include any or all of these protocols. Since the session layer is application protocol agnostic, it's transparent to layer 7 applications. This is similar to how Stunnel [27] or SSH tunnels [28] work. Those tools, of course, have few ambitions beyond providing transparent encryption.

Authentication and access control on a network level is currently a problem solved by a combination of VPNs [12] and firewalls [7]. Surprisingly, existing networks have made these technologies work from time to time, but it seems not unreasonable to point out that in practice networks experience problems using access control technologies which only make decisions based on what number a user's computer currently is assigned by their network. A VPN exists on most networks to allow an end-user to borrow a number from another network when the one their computer has doesn't allow them to access the resources they want. *Fived* on the other hand, ties traffic streams directly to a user's identity and uses that to make access control decisions.

Network mobility is a feature which allows a device to switch underlying network transports without breaking their network connections. This feature is used today in cell phone networks [15] where devices roam between cells routinely and so mobility is built into the low-level network protocol. OpenFlow [18] is another system which has mobility features, allowing devices to move network flows from endpoint to endpoint. Both require extensive levels of support in the networking hardware. *Fived* implements these features in the session layer.

Stream multiplexing has become common again with the introduction of HTTP/2.0 [4] which formalized SPDY's [3] approach of multiplexing multiple HTTP streams over a single TCP connection. *Fived*'s multiplexing is somewhat different, since it is protocol agnostic. This allows any traffic between two endpoints to share a transport stream.

Virtual hosts allows application protocols servers to host more than one hostname on the same IP address. [16] This technique is present in some application protocols, like HTTP and SMTP, but is not uniformly deployed through the network. Generally, when an application connects to a port on a host, the service is not given hostname information to allow it to appropriately determine which content to send. *Fived* introduces protocol-agnostic virtual hosting by enabling the hostname the user specifies to alter service routing in the session server.

Distributed identity has increased in prevalence since the launch of OpenID [21] a decade ago. Since then, many large web companies have shipped their own incompatible distributed identity systems, from Facebook [10] to Twitter [26] to Google [13] there's as many different identity protocols as there are companies that want to control identity information. *Fived* also includes a light-weight distributed identity protocol, which allows users to use an authenticated session to prove their identity to third-parties. One difference with *fived*'s protocol however, is that it eliminates the direct communication between the third party identity consumer and the identity provider, thus allowing for distributed identities that don't require users reveal to the identity provider where they're using the identity.

3 Technical Detail

3.1 Core Services

The following core services are the primitives I've selected to put into *fived*'s default set. *Fived*'s session layer protocol is loosely derived from the *tcpmux* protocol specified in RFC 1078 [17], from 1988. The basic *tcpmux* protocol is simple and can be implemented in under 100 lines of C. Each of the core services *fived* adds take anywhere from tens of lines of code to several hundred lines of code. These services work together to provide a broad range of session services. The essential features include service multiplexing, role-based authenticated access control, transparent session-wide encryption, mobility, virtual hosting and distributed identities.

Let's examine each of the core services in depth:

3.1.1 LIST

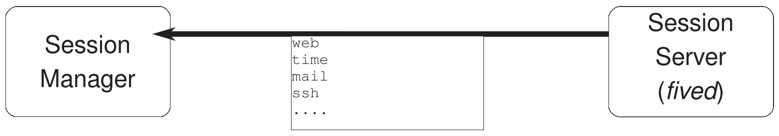

Figure 1: LIST Command

LIST enabled service discovery. LIST outputs a multi-line message which must be a list of the service names of the supported services, one name per line.[17] "Supported services" means services which the user is able to access. Services with restrictions only appear in LIST after a user has authenticated themselves with an authorized set of credentials. When LIST is followed by a service name, *fived* returns the service name if it exists. These two modes allow for dynamic service and extension discovery.

> **≫LIST≪**
> LIST
> MULTIPLEX
> TLS
> HOST
> http

Example Usage: A typical use of the LIST command on a *fived* daemon which supports a the core features LIST, MULTIPLEX, TLS and HOST as well as a service called "http."

3.1.2 AUTH

AUTH allows a user to authenticate a session. The exact mechanism to do this is server-specific as each organization tends to have their own requirements for credentialing users. The current *fived* prototype uses the Pluggable Authentication Modules (PAM) [2] system in place on most UNIX machines. The AUTH service negotiates authentication technologies and proceeds to engage the client in a mutually agreed challenge/response protocol. When the back-end authentication service is satisfied of the client's identity, the AUTH service relays the results to the client and attaches an identity to the session. After a session has been granted a certain identity, they may be authorized to access restricted services or other resources. The *fived* daemon can also include a mechanism to

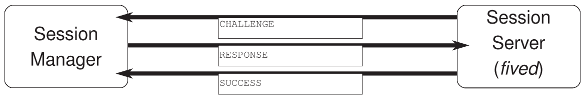

Figure 2: AUTH Command

pass the session's authenticated identity information through to underlying services they connect to.

Finally, depending on the service provider's setup, they may not wish this service be supported until after the user gains a secure channel for their session using the TLS service which is described in the following subsection. In this case, AUTH itself acts as a restricted service until after TLS or another acceptable encryption scheme is invoked.

3.1.3 TLS

TLS allows for session encryption. As the name might indicate, the TLS service is a command to the session server to start a TLS handshake. After the client requests this service, the session server and the client immediately engage in a TLS handshake and set up a secure channel. The client should retain the certificate offered by the server for future connections to ensure security. This is similar to how SSH handles key verification and has been moderately more successful than the web-based model for TLS. However, the client should feel free to use other mechanisms, such as the existing TLS PKI, to verify the certificate during first connect. After both the client and the server has completed the TLS handshake, the session continues over an encrypted channel. In addition, the server may choose to authenticate the client on the basis of a client-side certificate they present during the handshake.

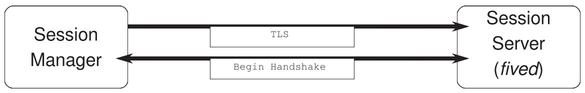

Figure 3: TLS Command

Here is an example of a user using the TLS service, then authenticating using AUTH and receiving access to the service *telnet* which they then access over a secure and authenticated session:

```
≫LIST≪
TLS
AUTH
≫TLS≪
+ SUCCESS
a TLS handshake takes place and the session continues over a secure channel:
≫AUTH <up,ext,pubkey>≪
+ SUCCESS <up>
Enter Username: ≫researcher≪
Enter Password: ≫secret≪
Authentication as researcher successful
≫LIST≪
telnet
research-service
≫telnet≪
researcher@researchbox  $
```

3.1.4 MULTIPLEX

MULTIPLEX allows more than one service on a session. In the basic protocol, when a client requests a service, the connection is taken over by the service and there is no further interaction with *fived*. Multiplexing allows a user to connect to a session server, encrypt their session, authenticate and then access as many services as they need.

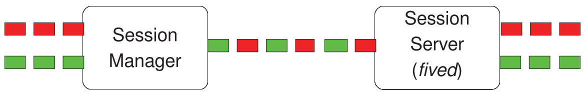

Figure 4: MULTIPLEX Command

Figure 4 shows a multiplexed session where multiple services are interacting with multiple clients. The client computer runs a session manager that handles the client connections from that machine while the service provider runs services through the *fived* daemon. There is no requirement that the session manager and the clients be on the same machine, nor is there a requirement that the session server and service daemons be on the same machine. This network-transparent interaction allows for organizations to create unified session servers that are the frontend for all of that organization's services. This allows for centralized authentication and also could allow a session server to act as a load balancer for various backend services.

In response to the MULTIPLEX command, *fived* begins the session multiplexing protocol. To multiplex more than one application layer datastream on top of the same reliable bytestream, *fived* uses a series of headers to delineate datastreams. Figure 5 shows the header format:

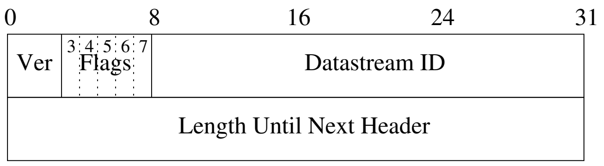

Figure 5: MULTIPLEX header format

The header fits in 64 bits, which allows for easy manipulation on most modern processing units. The first 3 bits comprise a version number, the next 5 bits contain flags, the subsequent 24 bits contain the datastream ID which identifies the datastream which follows the header and the final 32 bits is the length, in octets, until the next header. The meaning of the flags are as follows:

- **Bits 3 & 4** – *Reserved for future use.*
- **Bit 5** – *Complete* – This flag is set for a one-sided close in a duplex transport protocol. (As in `shutdown()` in the standard sockets interface.) The side that sends this flag is declaring that they no longer will be *sending* data. The datastream ID is still active, unless or until the other side sends a message with the complete or close flag. The length to next header field must be set to zero when this flag is set. (This header may not proceed data using this datastream ID.)
- **Bit 6** – *Close* – This flag is set when the application using this datastream is no longer willing to communicate. The other side should discontinue use. Any data for this ID will be dropped. The length to next header field must be zero when this flag is set.
- **Bit 7** – *New* – This flag is set by the server when the user asks to use a new service. The datastream ID will be new and identifies data from that service from now on. The datastream ID of zero is reserved for talking to the *fived* daemon.

This multiplexing protocol is sufficient for a user to access multiple services concurrently using their session. Their authentication stays intact and the encryption continues. The session persists until the user closes their connection to the session daemon.

3.1.5 HOST

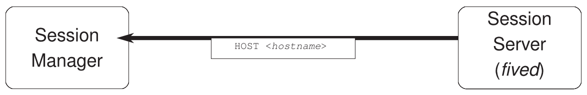

Figure 6: HOST Command

HOST enables a session server to provide services for many hosts, virtual or physical. Depending on the host selected, different services can be enabled. When a client issues the HOST command they provide a hostname or service-group name. Assuming the session's privilege level is sufficient and the name the client requests exists, the daemon issues an affirmative response and associates the session with the requested hostname. This allows the session layer to do virtual hosting at a network level. This allows organizations to centralize sessions into a small set of session servers which act much like load balancers do in existing networks.

3.1.6 DETACHABLE

DETACHABLE allows a client to disconnect from a session without destroying its state. If allowed by the server, DETACHABLE is a mechanism to request the server maintain a session's state while a client disconnects from the server for a time. This is almost a network equivalent of the UNIX screen [1] command. The DETACHABLE service provides the user with some sort of secret. This secret could be a cryptographic certificate, a password, ASCII art or any piece of data appropriate for the security requirements of the session. When the client disconnects from the session server, the session's state persists. Data from services which remain open will be queued. The amount of time a session's state is perserved and the amount of traffic it is willing to queue is up to the administrator of the session server.

3.1.7 ATTACH

ATTACH allows a client to resume a previously detached session. The user provides the secret issued by a previous invocation of the DETACHABLE service along with the number of bytes they've received since the session began. After verifying the secret, the user will be allowed to resume their session. However, since the user is likely to want to start a TLS session before providing the secret to the ATTACH service to prevent man in the middle attacks, resumed sessions will use this new TLS session, if one exists, instead of resuming an old one. (This also provides a re-key mechanism for long-standing sessions.)

3.1.8 Broader Uses of DETACHABLE and ATTACH

It is not required to break a session connection before using ATTACH on a DETACHABLE session. Instead, a user can attach another layer 4 connection to their existing layer 5 session. This allows different quality of service properties or connection bonding. DETACHABLE and ATTACH can also be used on one specific connection, which allows users to gracefully roam networks or even physical machines.

3.1.9 PROVEAUTH and GETSIGNKEY

This service provides a lightweight distributed identity system. PROVEAUTH allows a user to use their session to prove their identity to another system. Where AUTH creates a system of authentication for the session layer, PROVEAUTH allows a user to prove that identity elsewhere. This allows users to use an identity from one entity to authenticate with another.

In these types of protocols, there are three parties:

The identity provider (P) This is the entity providing the identity. It holds an authoritative notion of identity for its domain and chooses to grant these identities to users. In *fived* this entity is the session server providing the PROVEAUTH service.

The user (U) This is the end-user of the identity. In our session protocol, this is the user controlling the session client.

The identity consumer (C) This is a separate entity who accepts identities asserted by the identity provider and wishes to ensure that the user has a right to use a particular identity.

The protocol for a user to prove an identity to an identity consumer is as follows:

1. User (U) sends the identity to identity consumer (C) they want to prove is theirs.

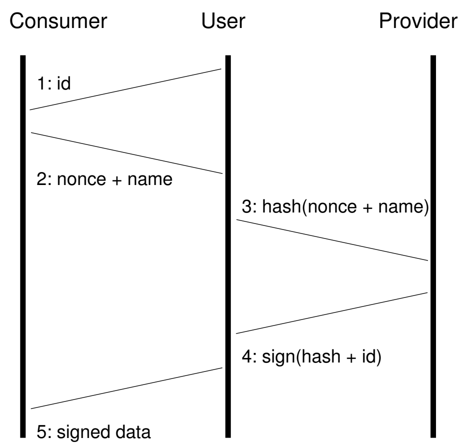

Figure 7: PROVEAUTH Protocol

2. Consumer (C) responds with a challenge to user (U). The challenge consists of a nonce chosen by C along with the canonical name for C.
3. User (U) concatenates the nonce and canonical names provided by the consumer (C) and hashes them. User (U) then invokes PROVEAUTH on their authenticated session with the identity provider (P) and provides this hash.
4. Provider (P) concatenates the hash with the identity the user's (U) session is authenticated as and signs the result using an RSA keypair whose public component is known by the consumer (C).
5. User (U) returns this signed data to the identity consumer (C).
6. The identity consumer (C) proves that user (U) has a right to the identity by verifying the signature on the data and ensuring the contents of the signed message matches the identity, nonce and canonical name expected.

This protocol allows for something many other distributed authentication protocols don't: it allows users to use their identities elsewhere without revealing who they pass their identity too. Unlike other major protocols (Facebook Connect, Twitter Auth, etc) where the person who controls your identity has a complete list of where you use it and when, this protocol omits the ability for identity providers to engage in that level of tracking.

GETSIGNKEY is a convenience service which offers the public key used to prove identities in PROVEAUTH. This service provides one mechanism out of many that identity consumers could receive the public keys for the signing keypairs for the identity providers they wish to support.

3.2 Session Initiator

Fived's session initiator has a larger goal of breaking network applications' dependence on the lower layers of the Internet. One of the major bottlenecks in the existing transition in-progress between IPv4 and IPv6 is that applications are required to be aware of IP addresses. This knowledge is necessary for applications even though users mostly specify computers by hostname. Yet, the application itself is responsible for the name resolution. Once it resolves the name, it must pass the correct layer 4 address to the underlying networking APIs.

The session initiator changes this. With it, the session API and session architecture take control earlier. The session initiator performs the initial connection establishment on behalf of the application. This allows applications using the session stack to move beyond the existing APIs focused on addresses and port numbers and simply ask the networking stack for a service. The session initiator needs to know two things: the name of the organization or computer the application would like to communicate with and the name of the service the application would like to access. With that, it does the rest and sets up the session.

Once applications move away from using addresses and port numbers, the underlying architecture of the Internet can evolve without nearly as much hassle. The session initiator will be the only thing that needs to change to allow applications to connect to each other in new ways. Arguably this only moves the problem around, but importantly it moves it to a place better designed for change, future expansion and alteration. The session layer is a more appropriate abstraction and interface for applications on the Internet.

4 Performance

Performance is always a critical issue. Users see performance overhead as a cost to almost any security technology. The cost users are willing to accept varies widely, but it seems fairly clear that the higher the performance cost, the more difficult adoption becomes.

The performance concerns for *fived* lie in two main areas:

- The increased cost of connection establishment with the session layer.
- The increased overhead during a data transfer across the network.

From the perspective of a user, these two things can be measured with two metrics. The first is the amount of time required from beginning a request until the first byte of data is available to the endpoint application. This is commonly referred to as "Time To First Byte" (TTFB) and generally establishes a lower bound on network latency. The second metric is the amount of time it takes for a request to *complete*. This is harder to establish for *fived* since a session layer is generic infrastructure that supports a variety of protocols with a variety of users and uses. There's no firm definition for the end of a request. So the metrics I used were Time to Thousandth Byte and Time to Millionth Byte, which roughly correspond to a small one kilobyte data transfer or a larger one megabyte data transfer across the network.

Experiments were conducted across the Internet using remote endpoints in two different cities. Average round-trip latency between the computers was 28.36 milliseconds with a standard deviation of 3.40 milliseconds. No significant packet loss was measured on the link. The server side computer was connected to the Internet on a university network which routes to the Internet via fiber, similar to most datacenter environments. The client-side computer was connected to the Internet on a lower bandwidth connection which is similar to most residential environments.

Measuring 500 datapoints shows the Layer 5 Time To First Byte is larger than the Layer 4 Time To First Byte, showing the expected performance degradation caused by needing to interact with a session server before being able to start an application protocol. While the difference is highly statistically significant, there is also overlap between the standard deviation of each data set as shown on the graph. Which means many individual uses of the session layer will not be significantly distinguishable from ordinary network jitter.

The story gets better when you look at time to completion. The gap between Layer 4 metrics and Layer 5 metrics narrows as more bytes are transferred across the connection, which shows

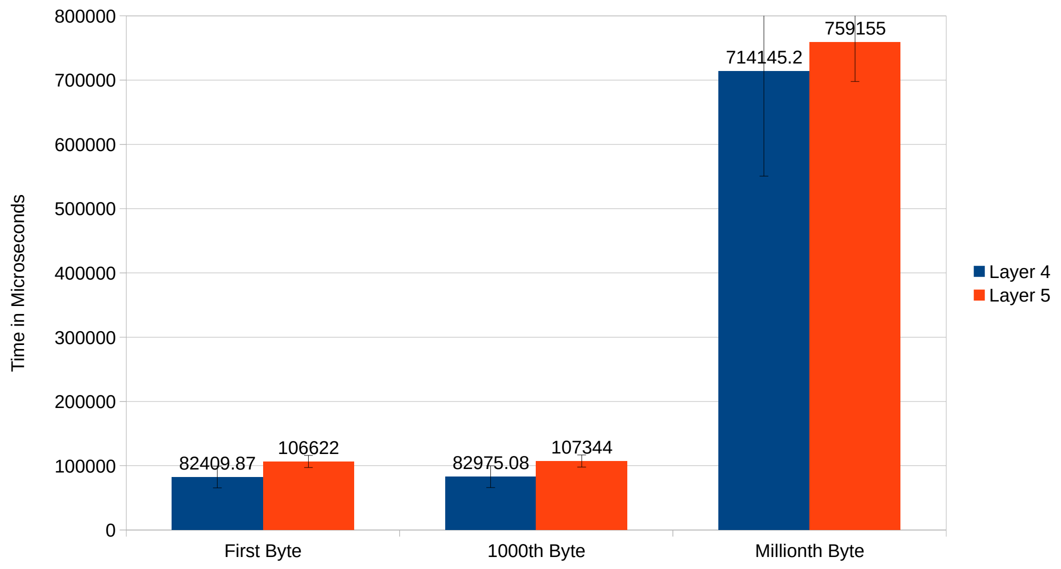

Figure 8: Layer 4 vs Layer 5 Performance

the dominating performance impact of the session layer is in the initial establishment of the session. While this doesn't show up as much with a short transfer of a thousand bytes of data, the performance gap at a million bytes of data is substantially lower.

For an implementation of *fived* entirely in userspace with no kernel components and several remaining optimization opportunities, this is not a particularly bad performance picture. It seems likely that the performance of *fived* may be manageable.

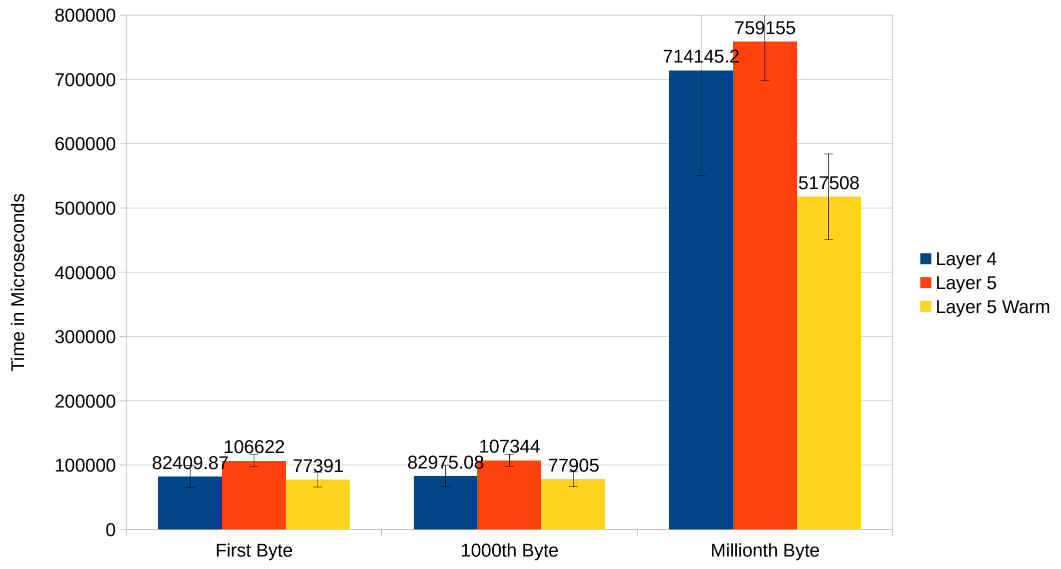

Figure 9: Layer 4 vs Layer 5 Performance with Warm Connections

Of course, this isn't the whole performance picture of *fived*. The opportunities of a session layer allow us flexibility application protocols otherwise don't have. The data we've gathered so far tells us the story of what happens when you need to get data from a peer across the Internet when you don't have a session established yet and need to set one up before being able to transfer data, but what of the cases where a session exists?

The same experiment from above was repeated with a "warm" connection where a session was already established. In this environment, we show that far from a negative performance impact, the session layer delivers a significant performance *improvement*. Not only during Time To First Byte connection establishment, but all the way through the millionth byte of data transferred. Avoiding TCP slow start appears to provide substantial benefit that endures through the transfer.

While this benefit isn't exclusive to *fived* and application protocols have re-designed themselves, sometimes substantially, to employ similar tricks (HTTP 1.1 essentially provided persistent connections and HTTP 2.0 essentially provides multiplexing) the session architecture allows these performance benefits to transparently apply to any protocol running on it. Instead of always imposing a cost, it's quite likely that the session layer can bring performance improvements and optimizations, possibly even tailored to the specific network environment for each computer, without re-writing every application which implements an application protocol.

5 Potential Improvements

5.1 Integration into the Kernel

Integrating some parts of the *fived* client into the kernel may provide considerable opportunities for performance enhancements resulting from less context switching, less copying between buffers and other clever opportunities that occur in kernelspace with full access to the kernel networking stack. The *fived* server could be similarly accelerated, though it seems fair to say that while demultiplexing might be suitable for inclusion into the kernel, a good portion of the logic in the *fived* server could remain in userspace where it can be customized and easily changed.

5.2 Integration into Hardware

For larger networks and integration of *fived* into switches and routers, it makes sense to develop specialized hardware. A considerable amount of the active work *fived* performs during most connections is reading the multiplexing headers and simply forwarding traffic. This could be very efficiently implemented in hardware.

Fortunately interacting with the *fived* server directly is a rare operation. It seems acceptable to expect that the core routing and forwarding portion use hardware acceleration, while interacting with the session layer (i.e. requesting new services) can be an exceptional operation handled outside of hardware or even by a separate machine. This machine then handles the request using higher level processing power and then hands down a new forwarding path for the accelerated multiplexer and demultiplexer to use when there's a new connection being established.

6 Deployment

Deployment needs to be a key concern with any new networking technology. The goal is to ensure that no network or set of users find themselves unable to benefit from *fived*. With the design of *fived*, an end-user can start gaining the benefits of a session layer if any of the following occur: 1) Their operating system vendor incorporates it into the networking environment for the operating system. 2) The network operator deploys session technology on their network. 3) The user runs any application that natively uses *fived*'s session layer. If **any** of these conditions apply, compatibility toolsets will allow most users to gain some of the key protections and benefits of *fived*. In this section, we talk about how *fived* can be successfully deployed in each of these three cases.

6.1 Deploying with Unmodified Applications

Naturally, today's applications do not already support *fived*. However, using a shared library preload unmodified applications can be retrofitted to use the session layer via a compatibility shim. The shim could intercept calls made to the networking interface, including name resolutions, sets up a session to the requested destination, and routes traffic through the session. These applications would then be able to transparently take advantage of a user's authentication credentials on a session or any increased access level, session mobility and reconnection features or transparent encryption services supported by the server. In addition, in the case that the application uses an deprecated connection protocol, the shim could convert its API calls into a request to the session initiator. This allows applications to use protocols that didn't exist at the time the application was written to reach services.

In the case of an unmodified server application, no changes are required since the interactions servers have with the *fived* daemon appear no different than any other network connection. However, a user who wishes to *only* expose a service through *fived* will have to reconfigure their server to bind to a location on the machine only *fived* can access. The easiest way to do this is generally to configure servers to bind themselves to localhost or a local socket. It should be noted though, that users are free to expose services via *fived* session layer while still keeping them open to all non-session layer users via traditional means.

With *fived* users are not forced to take an all or nothing approach, the migration to a session layer can happen slowly. *Private* services that aren't intended to be visible to the entire world will generally be the easiest to migrate. Since many private services are offered by organizations to people affiliated with them, it's easier for these organizations to require people install a *fived* client. For public-facing services, it will take time to get to a point where all users have migrated and support for direct layer 4 connections can be disabled on legacy services.

6.2 Deploying with Unmodified Computers

Another compatibility option allows a network administrator to session-enable segments of their network without waiting for each of the individual hosts to get their own native session support. In this scheme, a DNS proxy (or similar technology) can point outgoing name resolutions to a session server. This session server can initiate sessions to the hosts requested. This could allow for the users on the network to access things they wouldn't otherwise be able to reach in the case that the network has an established authenticated session to some other organization, or it could merely ensure traffic traverses a wide area network over an encrypted link. For mobile vehicles, like buses, boats or airplanes, networks could also use the mobility features to maintain connections as the vehicle roams between network points, possibly reconnecting from an IP in a completely different Autonomous System.

A similar system could be built at the network segment perimeter without a cooperating DNS server. If the session client is placed along the route for outbound traffic, it is free to examine the destination of the packets coming from the unmodified host, open a session to that destination and tunnel the traffic over the session layer. Whether or not DNS spoofing provides network administrators and users with a more desirable solution is an open question.

7 Project Status

In this report we outline a more feature rich interface to the Internet. In the intervening 30 years since the concept of a session layer was first proposed, we've seen an expansion in the diversity of application needs on computer networks. We've seen in detail the plethora of mechanisms, workarounds and ad-hoc solutions networks have implemented on top of transport protocols to achieve these needs. *Fived* is an attempt to coalesce this set of needs in a unified and backwards compatible interface.

The prototype tested in this report continues to be under development. The current codebase is unoptimized and runs entirely in userspace on unmodified Linux kernels. This report outlines the current status of our session layer design and contains preliminary data to support the project's feasibility.

8 Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1018928. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Bibliography

[1] GNU Screen Project. [Online]. Available: http://www.gnu.org/software/screen/

[2] Linux PAM. [Online]. Available: http://www.kernel.org/pub/linux/libs/pam/

[3] M. Belshe and R. Peon, “draft-mbelshe-httpbis-spdy-00: SPDY protocol,” 2012.

[4] M. Belshe, M. Thomson, and R. Peon, “RFC 7540: Hypertext Transfer Protocol Version 2 (HTTP/2),” 2015.

[5] D. Bernstein. (2011) CurveCP: Usable security for the Internet. [Online]. Available: http://curvecp.org/

[6] J. Chandrashekar, Z. Zhang, Z. Duan, and Y. Hou, “Service oriented internet,” *Service-Oriented Computing-ICSOC 2003*, pp. 543–558.

[7] D. Chapman, E. Zwicky, and D. Russell, *Building internet firewalls.* O’Reilly & Associates, Inc. Sebastopol, CA, USA, 1995.

[8] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, “Planetlab: an overlay testbed for broad-coverage services,” *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, p. 12, 2003.

[9] T. Dierks and E. Rescorla, “RFC 5246: The transport layer security (TLS) protocol version 1.2,” Tech. Rep., 2008.

[10] Facebook Inc. Facebook Login. [Online]. Available: https://developers.facebook.com/products/login/

[11] A. Freier, P. Karlton, and P. Kocher, “Secure Socket Layer 3.0,” *IETF draft, November*, 1996.

[12] B. Gleeson, A. Lin, J. Heinanen, G. Armitage, and A. Malis, “RFC 2764: A Framework for IP Based Virtual Private Networks,” 2000.

[13] Google Inc. Google Sign-In for Websites. [Online]. Available: https://developers.google.com/identity/sign-in/web/

[14] Internet Corporation for Assigned Names and Numbers. (2014) Remaining IPv4 Addresses to be Redistributed to Regional Internet Registries. [Online]. Available: https://www.icann.org/news/announcement-2-2014-05-20-en

[15] R. Kwan, R. Arnott, R. Paterson, R. Trivisonno, and M. Kubota, “On mobility load balancing for LTE systems,” in *Vehicular Technology Conference, 1988, IEEE 38th*, 2010, pp. 1–5.

[16] B. Laurie and P. Laurie, *Apache: The definitive guide.* "O'Reilly Media, Inc.", 2003.

[17] M. Lotter. (1988, November) RFC 1078: TCP Port Service Multiplexer (TCPMUX). [Online]. Available: http://www.rfc-editor.org/rfc/rfc1078.txt

[18] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[19] L. Peterson, T. Anderson, D. Blumenthal, D. Casey, D. Clark, D. Estrin, J. Evans, D. Raychaudhuri, M. Reiter, J. Rexford *et al.*, "GENI design principles," *IEEE Computer*, vol. 39, no. 9, pp. 102–105, 2006.

[20] W. M. Petullo, X. Zhang, J. A. Solworth, D. J. Bernstein, and T. Lange, "MinimaLT: minimal-latency networking through better security," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security.* ACM, 2013, pp. 425–438.

[21] D. Recordon and B. Fitzpatrick, "OpenID Authentication 1.1," *Finalized OpenID Specification, May*, 2006.

[22] J. Saltzer, D. Reed, and D. Clark, "End-to-end arguments in system design," *ACM Transactions on Computer Systems (TOCS)*, vol. 2, no. 4, p. 288, 1984.

[23] R. Srinivasan, "RFC 1833: Binding protocols for onc RPC version 2," 1995.

[24] P. Staubach, B. Pawlowski, and B. Callaghan, "RFC 1813: NFS Version 3 Protocol Specification," 1995.

[25] The Chromium Project. QUIC, a multiplexed stream transport over UDP. [Online]. Available: https://www.chromium.org/quic

[26] Twitter Inc. Sign in with Twitter. [Online]. Available: https://dev.twitter.com/web/sign-in

[27] W. Wong, "Stunnel: SSLing Internet Services Easily," *SANS Institute, November*, 2001.

[28] T. Ylonen and C. Lonvick, "RFC 4254: The Secure Shell (SSH) Connection Protocol," 2006.