Ananta: Cloud Scale Load Balancing

Parveen Patel, Deepak Bansal, Lihua Yuan, Ashwin Murthy, Albert Greenberg,
David A. Maltz, Randy Kern, Hemant Kumar, Marios Zikos, Hongyu Wu,
Changhoon Kim, Naveen Karri

Microsoft

ABSTRACT

Layer-4 load balancing is fundamental to creating scale-out web services. We designed and implemented Ananta, a scale-out layer-4 load balancer that runs on commodity hardware and meets the performance, reliability and operational requirements of multi-tenant cloud computing environments. Ananta combines existing techniques in routing and distributed systems in a unique way and splits the components of a load balancer into a consensus-based reliable control plane and a decentralized scale-out data plane. A key component of Ananta is an agent in every host that can take over the packet modification function from the load balancer, thereby enabling the load balancer to naturally scale with the size of the data center. Due to its distributed architecture, Ananta provides direct server return (DSR) and network address translation (NAT) capabilities across layer-2 boundaries. Multiple instances of Ananta have been deployed in the Windows Azure public cloud with combined bandwidth capacity exceeding 1Tbps. It is serving traffic needs of a diverse set of tenants, including the blob, table and relational storage services. With its scale-out data plane we can easily achieve more than 100Gbps throughput for a single public IP address. In this paper, we describe the requirements of a cloud-scale load balancer, the design of Ananta and lessons learnt from its implementation and operation in the Windows Azure public cloud.

Categories and Subject Descriptors: C.2.4 [Computer-Communication Networks]: Distributed Systems

General Terms: Design, Performance, Reliability

Keywords: Software Defined Networking; Distributed Systems; Server Load Balancing

1. INTRODUCTION

The rapid rise of cloud computing is driving demand for large scale multi-tenant clouds. A multi-tenant cloud environment hosts many different types of applications at a low cost while providing high uptime SLA — 99.9% or higher [3, 4, 19, 10]. A multi-tenant load balancer service is a fundamental building block of such multi-tenant cloud environments. It is involved in almost all external and half of intra-DC traffic (§2) and hence its uptime requirements need to be at least as high as applications' SLA, but often significantly higher to account for failures in other infrastructure services.

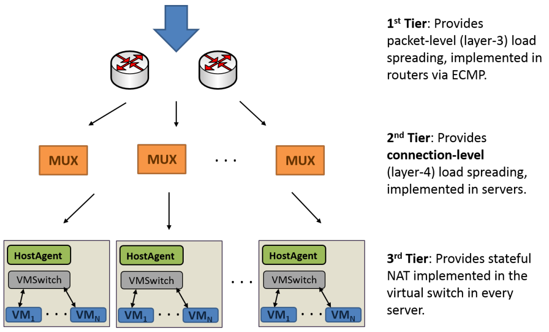

MUX
MUX
MUX
HostAgent
VMSwitch
VM_1
VM_N
HostAgent
VMSwitch
VM_1
VM_N
HostAgent
VMSwitch
VM_1
VM_N
1st Tier: Provides packet-level (layer-3) load spreading, implemented in routers via ECMP.
2nd Tier: Provides connection-level (layer-4) load spreading, implemented in servers.
3rd Tier: Provides stateful NAT implemented in the virtual switch in every server.

Figure 1: Ananta Data Plane Tiers

As a cloud provider, we have seen that cloud services put huge pressure on the load balancer's *control plane* and *data plane*. Inbound flows can be intense — greater than 100 Gbps for a single IP address — with every packet hitting the load balancer. The pay-as-you-go model and large tenant deployments put extremely high demands on real-time load balancer configuration — in a typical environment of 1000 hosts we see six configuration operations per minute on average, peaking at one operation per second. In our experience, the data plane and control plane demands drove our hardware load balancer solution into an untenable corner of the design space, with high cost, with SLA violations and with load balancing device failures accounting for 37% of all live site incidents.

The design proposed in this paper, which we call *Ananta* (meaning infinite in Sanskrit), resulted from examining the basic requirements, and taking an altogether different approach. Ananta is a scalable software load balancer and NAT that is optimized for multi-tenant clouds. It achieves scale, reliability and *any service anywhere* (§2) via a novel division of the data plane functionality into three separate tiers. As shown in Figure 1, at the topmost tier routers provide load distribution at the network layer (layer-3) based on the *Equal Cost Multi Path* (ECMP) routing protocol [25]. At the second tier, a scalable set of dedicated servers for load balancing, called *multiplexers* (Mux), maintain connection flow state in memory and do layer-4 load distribution to application servers. A third tier present in the virtual switch on every server provides stateful NAT functionality. Using this design, no outbound trafffic has to pass through the Mux, thereby significantly reducing packet processing requirement. Another key element of this design is the ability to offload multiplexer functionality down to the host. As discussed in §2, this design enables greater than 80% of the load balanced traffic to bypass the load balancer and go direct, thereby

SIGCOMM'13, August 12–16, 2013, Hong Kong, China.
Copyright 2013 ACM 978-1-4503-2056-6/13/08 ...$15.00.

eliminating throughput bottleneck and reducing latency. This division of data plane scales naturally with the size of the network and introduces minimal bottlenecks along the path.

Ananta's approach is an example of Software Defined Networking (SDN) as it uses the same architectural principle of managing a flexible data plane via a centralized control plane. The controller maintains high availability via state replication based on the Paxos [14] distributed consensus protocol. The controller also implements real-time port allocation for outbound NAT, also known as Source NAT (SNAT).

Ananta has been implemented as a service in the Windows Azure cloud platform. We considered implementing Ananta functionality in hardware. However, with this initial Ananta version in software, we were able to rapidly explore various options in production and determine what functions should be built into hardware, e.g., we realized that keeping per-connection state is necessary to maintain application uptime due to the dynamic nature of the cloud. Similarly, *weighted random* load balancing policy, which reduces the need for per-flow state synchronization among load balancer instances, is sufficient for typical cloud workloads. We consider the evaluation of these mechanisms, regardless of how they are implemented, to be a key contribution of this work.

More than 100 instances of Ananta have been deployed in Windows Azure since September 2011 with a combined capacity of 1Tbps. It has been serving 100,000 VIPs with varying workloads. It has proven very effective against DoS attacks and minimized disruption due to abusive tenants. Compared to the previous solution, Ananta costs one order of magnitude less; and provides a more scalable, flexible, reliable and secure solution overall.

There has been significant interest in moving middlebox functionality to software running on general-purpose hardware in both research [23, 24, 5] and industry [8, 2, 27, 21, 31]. Most of these architectures propose using either DNS or OpenFlow-enabled hardware switches for scaling. To the best of our knowledge Ananta is the first middlebox architecture that refactors the middlebox functionality and moves parts of it to the host thereby enabling use of network routing technologies — ECMP and BGP — for natural scaling with the size of the network. The main contributions of this paper to the research community are:

- Identifying the requirements and design space for a cloud-scale solution for layer-4 load balancing.
- Providing design, implementation and evaluation of Ananta that combines techniques in networking and distributed systems to refactor load balancer functionality in a novel way to meet scale, performance and reliability requirements.
- Providing measurements and insights from running Ananta in a large operational Cloud.

2. BACKGROUND

In this section, we first consider our data center network architecture and the nature of traffic that is serviced by the load balancer. We then derive a set of requirements for the load balancer.

2.1 Data Center Network

Figure 2 shows the network of a typical data center in our cloud. A medium sized data center hosts $40,000$ servers, each with one 10Gbps NIC. This two-level Clos network architecture [11] typically has an over-subscription ratio of $1:4$ at the spine layer. The border routers provide a combined capacity of 400Gbps for connectivity to other data centers and the Internet. A cloud controller manages resources in the data center and hosts *services*. A service is a collection of virtual or native machines that is managed as one entity. We use the terms *tenant* and *service* interchangeably in this paper. Each machine is assigned a private *Direct IP* (DIP) address. Typically, a service is assigned one public *Virtual IP* (VIP) address and all traffic crossing the service boundary, e.g., to the Internet or to back-end services within the same data center such as storage, uses the VIP address. A service exposes zero or more external *endpoints* that each receive inbound traffic on a specific protocol and port on the VIP. Traffic directed to an external endpoint is load-balanced to one or more machines of the service. All outbound traffic originating from a service is *Source NAT'ed* (SNAT) using the same VIP address as well. Using the same VIP for all inter-service traffic has two important benefits. First, it enables easy upgrade and disaster recovery of services since the VIP can be dynamically mapped to another instance of the service. Second, it makes ACL management easier since the ACLs can be expressed in terms of VIPs and hence do not change as services scale up or down or get redeployed.

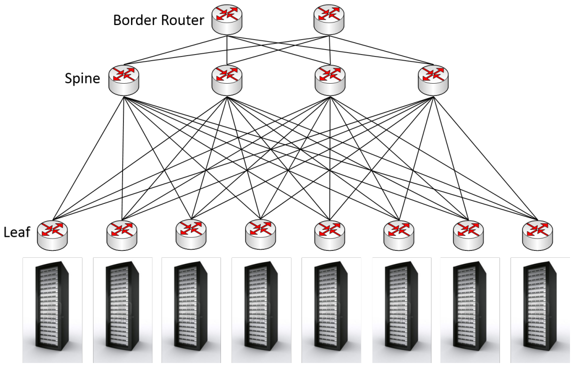


Figure 2: Flat Data Center Network of the Cloud. *All network devices run as Layer-3 devices causing all traffic external to a rack to be routed. All inter-service traffic — intra-DC, inter-DC and Internet — goes via the load balancer.*

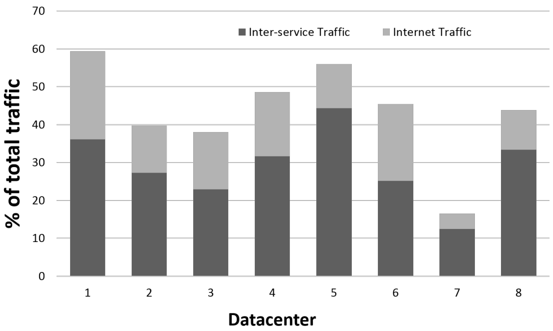


Figure 3: Internet and inter-service traffic as a percentage of total traffic in eight data centers.

2.2 Nature of VIP Traffic

Public cloud services [3, 4, 19] host a diverse set of workloads, such as storage, web and data analysis. In addition, there are an increasing number of third-party services available in the cloud. This trend leads us to believe that the amount of inter-service traffic in the cloud will continue to increase. We examined the total traffic in eight data centers for a period of one week and computed the ratio of Internet traffic and inter-service traffic to the total traffic.

The result is shown in Figure 3. On average, about 44% (with a minimum of 18% and maximum of 59%) of the total traffic is VIP traffic — it either needs load balancing or SNAT or both. Out of this, about 14% traffic on average is to the Internet and the remaining 30% is intra-DC. The ratio of intra-DC VIP traffic to Internet VIP traffic is $2 : 1$. Overall, we find that 70% of total VIP traffic is inter-service within the same data center. We further found that on average the ratio of inbound traffic to outbound traffic across our data centers is $1 : 1$. Majority of this traffic is read-write traffic and cross-data center replication traffic to our storage services. In summary, greater than 80% of VIP traffic is either outbound or contained within the data center. As we show in this paper, Ananta offloads all of this traffic to the host, thereby handling only 20% of the total VIP traffic.

2.3 Requirements

Scale, Scale and Scale: The most stringent scale requirements can be derived by assuming that *all* traffic in the network is either load balanced or NAT'ed. For a $40,000$ server network, built using the architecture shown in Figure 2, 400Gbps of external traffic and 100 Tbps of intra-DC traffic will need load balancing or NAT. Based on the traffic ratios presented in §2.2, at 100% network utilization, 44Tbps traffic will be VIP traffic. A truly scalable load balancer architecture would support this requirement while maintaining *low* cost. While cost is a subjective metric, in our cloud, less than 1% of the total server cost would be considered low cost, so any solution that would cost more than 400 general-purpose servers is too expensive. At the current typical price of US$2500 per server, the total cost should be less than US$1,000,000. Traditional hardware load balancers do not meet this requirement as their typical list price is US$80,000 for 20Gbps capacity without considering bulk discounts, support costs or redundancy requirements.

There are two more dimensions to the scale requirement. First, the bandwidth and number of connections served by a single VIP are highly variable and may go up to 100Gbps and 1million simultaneous connections. Second, the rate of change in VIP configuration tends to be very large and bursty, on average 12000 configurations per day for a 1000 node cluster, with bursts of 100s of changes per minute as customer services get deployed, deleted or migrated.

Reliability: The load balancer is a critical component to meet the uptime SLA of applications. Services rely on the load balancer to monitor health of their instances and maintain availability during planned and unplanned downtime. Over many years of operation of our cloud we found that traditional $1 + 1$ redundancy solutions deployed as active/standby hardware load balancers are unable to meet these high availability needs. The load balancer must support $N + 1$ redundancy model with auto-recovery, and the load balancing service must degrade gracefully in the face of failures.

Any Service Anywhere: In our cloud, applications are generally spread over multiple layer-2 domains and sometimes even span multiple data centers. The load balancer should be able to reach DIPs located anywhere on the network. Traditional load balancers provide some of their functionality, e.g., NAT, to DIPs only within a layer-2 domain. This fragments the load balancer capacity and makes them unusable in layer-3 based architectures.

Tenant Isolation: A multi-tenant load balancer is shared by thousands of services. It is critical that DoS attacks on one service do not affect the availability of other services. Similarly, an abusive service should not be able to affect the availability of NAT for other services by creating large number of outbound connections. In our cloud environment, we often see services with large number of outbound NAT connections due to bugs and poor application design. Furthermore, when the load balancer is under load, it is important that the load balancer provides each service its fair share of resources.

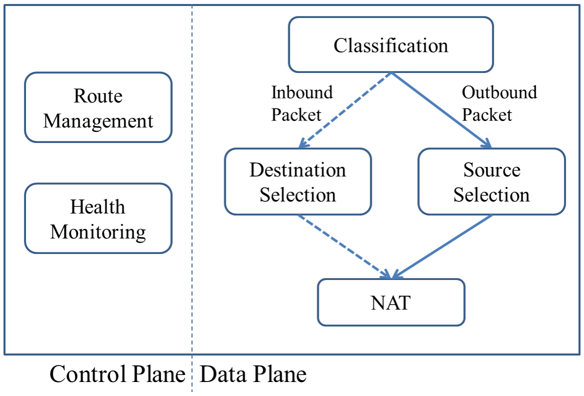


Figure 4: Components of a traditional load balancer. *Typically, the load balancer is deployed in an active-standby configuration. The route management component ensures that the currently active instance handles all the traffic.*

3. DESIGN

3.1 Design Principles

Scale-out In-network Processing: Figure 4 illustrates the main components of a traditional load balancer. For each new flow, the load balancer selects a destination address (or source for SNAT) depending on the currently active flows and remembers that decision in a flow table. Subsequent packets for that flow use the state created by the first packet. Traditional NAT and load balancing algorithms (e.g., round-robin) require knowledge of all active flows, hence all traffic for a VIP must pass through the same load balancer. This forces the load balancer into a *scale-up* model. A *scale-up* or *vertical scaling* model is one where handling more bandwidth for a VIP requires a higher capacity box.

Network routers, on the other hand, follow a *scale-out* model. A *scale-out* or *horizontal scaling* model is one where more bandwidth can be handled by simply adding more devices of similar capacity. Routers scale out because they do not maintain any per-flow state that needs synchronization across routers and therefore one can add or remove additional routers easily. Ananta design reduces the in-network functionality needed for load balancing to be such that multiple network elements can simultaneously process packets for the same VIP without requiring per-flow state synchronization.

This design choice is enabled because we can make certain assumptions about our environment. One of the key assumptions is that load balancing policies that require global knowledge, e.g., weighted round robin (WRR), are not required for layer-4 load balancing. Instead, randomly distributing connections across servers based on their weights is a reasonable substitute for WRR. In fact, *weighted random* is the only load balancing policy used by our load balancer in production. The weights are derived based on the size of the VM or other capacity metrics.

Offload to End Systems: Hypervisors in end systems can already do highly scalable network processing, e.g., ACL enforcement, rate limiting and metering. Ananta leverages this distributed scalable platform and offloads significant data plane and control plane functionality down to the hypervisor in end systems. The hypervisor needs to handle state only for the VMs hosted on it. This design choice is another key differentiator from existing load bal-

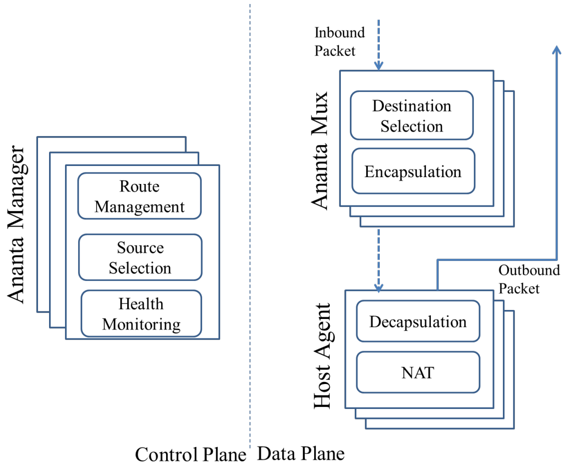


Figure 5: The Ananta Architecture. *Ananta consists of three components — Ananta Manager, Ananta Mux and Host Agent. Each component is independently scalable. Manager coordinates state across Agents and Muxes. Mux is responsible for packet forwarding for inbound packets. Agent implements NAT, which allows all outbound traffic to bypass Mux. Agents are co-located with destination servers.*

ancers. While on one hand it enables natural scaling with the size of the data center; on the other hand, it presents significant challenges in managing distributed state across all hosts and maintaining availability during failures of centralized components.

3.2 Architecture

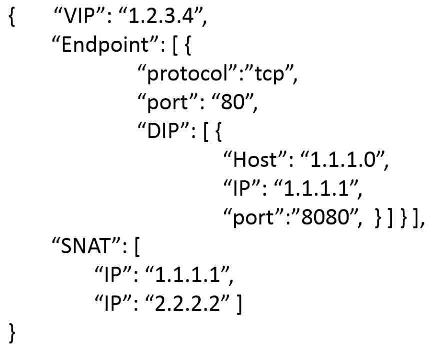
Ananta is a loosely coupled distributed system comprising three main components (see Figure 5) — *Ananta Manager* (AM), *Multiplexer* (Mux) and *Host Agent* (HA). To better understand the details of these components, we first discuss the load balancer configuration and the overall packet flow. All packet flows are described using TCP connections but the same logic is applied for UDP and other protocols using the notion of *pseudo connections*.

3.2.1 VIP Configuration

The load balancer receives a VIP Configuration for every VIP that it is doing load balancing and NAT for. A simplified VIP configuration is shown in Figure 6. An *Endpoint* refers to a specific transport protocol and port on the VIP that is load balanced to a set of *DIPs*. Packets destined to an Endpoint are NAT'ed to the DIP address and port. *SNAT* specifies a list of IP addresses for which outbound connections need to be Source NAT'ed with the VIP and an ephemeral port.

3.2.2 Inbound Connections

Figure 7 shows how packets destined for a VIP are load balanced and delivered to the DIP of a VM. When a VIP is configured on Ananta, each Mux advertises a route to its first-hop router announcing itself as the next hop for that VIP[1]. This causes the routers to distribute packets destined for the VIP across all the Mux nodes based on Equal Cost MultiPath Routing Protocol (ECMP) [25] (step 1). Upon receiving a packet, the Mux chooses a DIP for the connection based on its load balancing algorithm, described later in this section. It then encapsulates the received packet using IP-in-IP protocol [18] setting the selected DIP as the destination address in the outer header (step 2). It then sends it out using regular IP routing at the Mux (step 3). The Mux and the DIP do not need to be on the same VLAN, they just need to have IP (layer-3) connectivity between them. The HA, located on the same physical machine as the target DIP, intercepts this encapsulated packet, removes the outer header, and rewrites the destination address and port (step 4) and remembers this NAT state. The HA then sends the rewritten packet to the VM (step 5).



```
{     "VIP": "1.2.3.4",
      "Endpoint": [ {
                  "protocol":"tcp",
                  "port": "80",
                  "DIP": [ {
                              "Host": "1.1.1.0",
                              "IP": "1.1.1.1",
                              "port":"8080",  } ] } ],
      "SNAT": [
            "IP": "1.1.1.1",
            "IP": "2.2.2.2" ]
}
```

Figure 6: JSON representation of a simple VIP Configuration.

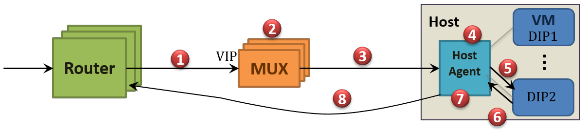


Figure 7: Load Balancing for Inbound Connections.

When the VM sends a reply packet for this connection, it is intercepted by the HA (step 6). The HA does a reverse NAT based on the state from step 4 and rewrites the source address and port (step 7). It then sends the packet out to the router towards the source of this connection. The return packet does not go through the Mux at all, thereby saving packet processing resources and network delay. This technique of bypassing the load balancer on the return path is known as Direct Server Return (DSR). Not all packets of a connection would end up at the same Mux, however all packets for a single connection must be delivered to the same DIP. Muxes achieve this via a combination of consistent hashing and state management as explained later in this section.

3.2.3 Outbound Connections

A unique feature of Ananta is a distributed NAT for outbound connections. Even for outbound connections that need source NAT (SNAT), Ananta ensures that outgoing packets do not need to go through Mux. Figure 8 shows how packets for an outbound SNAT

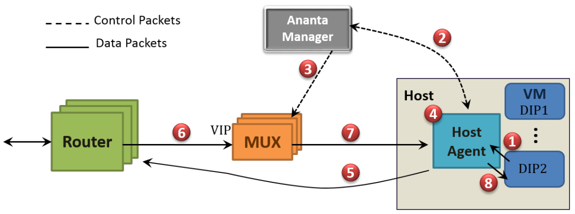


Figure 8: Handling Outbound SNAT Connections.

[1] In reality, routes are advertised for VIP subnets due to small routing tables in commodity routers but the same logic applies.

connection are handled. A VM sends a packet containing its DIP as the source address, $port_d$ as the port and an external address as the destination address (step 1). The HA intercepts this packet and recognizes that this packet needs SNAT. It then holds the packet in a queue and sends a message to AM requesting an externally routable VIP and a port for this connection (step 2). AM allocates a $(VIP, port_s)$ from a pool of available ports and configures each Mux with this allocation (step 3). AM then sends this allocation to the HA (step 4). The HA uses this allocation to rewrite the packet so that its source address and port are now $(VIP, port_s)$. The HA sends this rewritten packet directly to the router. The return packets from the external destination are handled similar to inbound connections. The return packet is sent by the router to one of the Mux nodes (step 6). The Mux already knows that $DIP2$ should receive this packet (based on the mapping in step 3), so it encapsulates the packet with $DIP2$ as the destination and sends it out (step 7). The HA intercepts the return packet, performs a reverse translation so that the packet's destination address and port are now $(DIP, port_d)$. The HA sends this packet to the VM (step 8).

3.2.4 Fastpath

In order to scale to the 100s of terabit bandwidth requirement of intra-DC traffic, Ananta offloads most of the intra-DC traffic to end systems. This is done by a technique we call *Fastpath*. The key idea is that the load balancer makes its decision about which DIP a new connection should go to when the first packet of that connection arrives. Once this decision is made for a connection it does not change. Therefore, this information can be sent to the HAs on the source and destination machines so that they can communicate directly. This results in the packets being delivered directly to the DIP, bypassing Mux in both directions, thereby enabling communication at full capacity supported by the underlying network. This change is transparent to both the source and destination VMs.

To illustrate how Fastpath works, consider two services 1 and 2 that have been assigned virtual addresses $VIP1$ and $VIP2$ respectively. These two services communicate with each other via $VIP1$ and $VIP2$ using the algorithms for load balancing and SNAT described above. Figure 9 shows a simplified version of packet flow for a connection initiated by a VM $DIP1$ (belonging to service 1) to $VIP2$. The source host of $DIP1$ SNATs the TCP SYN packet using $VIP1$ and sends it to $VIP2$ (step 1). This packet is delivered to a Mux2, which forwards the packet towards destination $DIP2$ (step 2). When $DIP2$ replies to this packet, it is SNAT'ed by the destination host using $VIP2$ and sent to Mux1 (step 3). This Mux uses its SNAT state and sends this packet to $DIP1$ (step 4). Subsequent packets for this connection follow the same path.

For Fastpath, Ananta configures Mux with a set of source and destination subnets that are capable of Fastpath. Once a connection has been fully established (e.g., TCP three-way handshake has completed) between $VIP1$ and $VIP2$, Mux2 sends a redirect message to $VIP1$, informing it that the connection is mapped to $DIP2$ (step 5). This redirect packet goes to a Mux handling $VIP1$, which looks up its table to know that this port is used by $DIP1$. Mux1 then sends a redirect message towards $DIP1$ and $DIP2$ (steps 6 and 7 respectively). HA on the source host intercepts this redirect packet and remembers that this connection should be sent directly to $DIP2$. Similarly HA on the destination host intercepts the redirect message and remembers that this connection should be sent to $DIP1$. Once this exchange is complete, any future packets for this connection are exchanged directly between the source and destination hosts (step 8).

There is one security concern associated with Fastpath – a rogue host could send a redirect message impersonating the Mux and hijack traffic. HA prevents this by validating that the source address of redirect message belongs to one of the Ananta services in the data center. This works in our environment since the hypervisor prevents IP spoofing. If IP spoofing cannot be prevented, a more dynamic security protocol such as IPSEC can be employed.

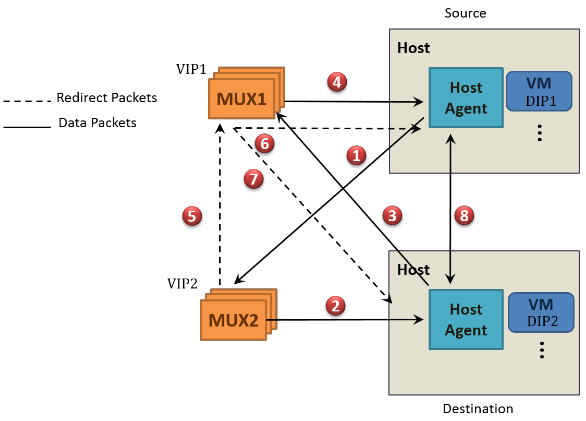


Figure 9: Fastpath Control and Data Packets. Routers are not shown for brevity. Starting with step 8, packets flow directly between source and destination hosts.

3.3 Mux

The Multiplexer (Mux) handles all incoming traffic. It is responsible for receiving traffic for all the configured VIPs from the router and forwarding it to appropriate DIPs. Each instance of Ananta has one or more sets of Muxes called *Mux Pool*. All Muxes in a Mux Pool have uniform machine capabilities and identical configuration, i.e., they handle the same set of VIPs. Having a notion of Mux Pool allows us to scale the number of Muxes (data plane) independent of the number of AM replicas (control plane).

3.3.1 Route Management

Each Mux is a BGP *speaker* [20]. When a VIP is configured on Ananta, each Mux starts advertising a route for that VIP to its first-hop router with itself as the next hop. All Muxes in a Mux Pool are equal number of Layer-3 network hops away from the entry point of the data center. This ensures that the routers distribute traffic for a given VIP equally across all Muxes via the Equal Cost MultiPath (ECMP) routing protocol [25]. Running the BGP protocol on the Mux provides automatic failure detection and recovery. If a Mux fails or shuts down unexpectedly, the router detects this failure via the BGP protocol and automatically stops sending traffic to that Mux. Similarly, when the Mux comes up and it has received state from AM, it can start announcing the routes and the router will start forwarding traffic to it. Muxes use the TCP MD5 [13] protocol for authenticating their BGP sessions.

3.3.2 Packet Handling

The Mux maintains a mapping table, called *VIP map*, that determines how incoming packets are handled. Each entry in the mapping table maps a VIP endpoint, i.e., three-tuple (VIP, IP protocol, port), to a list of DIPs. The mapping table is computed by AM and sent to all the Muxes in a Mux Pool. When Mux receives a packet from the router, it computes a hash using the five-tuple from packet header fields — (source IP, destination IP, IP Protocol, source port, destination port). It then uses this hash to lookup a DIP from the list of DIPs in the associated map. Finally, it encapsulates [18] the packet with an outer IP header — with itself as the source IP and the DIP as the destination IP — and forwards this packet to the DIP.

The encapsulation at the Mux preserves the original IP header and IP payload of the packet, which is essential for achieving Direct Server Return (DSR). All Muxes in a Mux Pool use the exact same hash function and seed value. Since all Muxes have the same mapping table, it doesn't matter which Mux a given new connection goes to, it will be directed to the same DIP.

3.3.3 Flow State Management

Mux supports two types of mapping entries – *stateful* and *stateless*. Stateful entries are used for load balancing and stateless entries are used for SNAT. For stateful mapping entries, once a Mux has chosen a DIP for a connection, it remembers that decision in a flow table. Every non-SYN TCP packet, and every packet for connection-less protocols, is matched against this flow table first, and if a match is found it is forwarded to the DIP from the flow table. This ensures that once a connection is directed to a DIP, it continues to go to that DIP despite changes in the list of DIPs in the mapping entry. If there is no match in the flow table, the packet is treated as a first packet of the connection.

Given that Muxes maintain per-flow state, it makes them vulnerable to state exhaustion attacks such as the SYN-flood attack. To counter this type of abuse, Mux classifies flows into *trusted* flows and *untrusted* flows. A trusted flow is one for which the Mux has seen more than one packet. These flows have a longer idle timeout. Untrusted flows are the ones for which the Mux has seen only one packet. These flows have a much shorter idle timeout. Trusted and untrusted flows are maintained in two separate queues and they have different memory quotas as well. Once a Mux has exhausted its memory quota, it stops creating new flow states and falls back to lookup in the mapping entry. This allows even an overloaded Mux to maintain VIP availability with a slightly degraded service.

3.3.4 Handling Mux Pool Changes

When a Mux in a Mux Pool goes down, routers take it out of rotation once BGP hold timer expires (we typically set hold timer to 30 seconds). When any change to the number of Muxes takes place, ongoing connections will get redistributed among the currently live Muxes based on the router's ECMP implementation. When this happens, connections that relied on the flow state on another Mux may now get misdirected to a wrong DIP if there has been a change in the mapping entry since the connection started. We have designed a mechanism to deal with this by replicating flow state on two Muxes using a DHT. The description of that design is outside the scope of this paper as we have chosen to not implement this mechanism yet in favor of reduced complexity and maintaining low latency. In addition, we have found that clients easily deal with occasional connectivity disruptions by retrying connections, which happen for various other reasons as well.

3.4 Host Agent

A differentiating component of the Ananta architecture is an agent, called Host Agent, which is present on the host partition of every physical machine that is served by Ananta. The Host Agent is the key to achieving DSR and SNAT across layer-2 domains. Furthermore, the Host Agent enables data plane scale by implementing Fastpath and NAT; and control plane scale by implementing VM health monitoring.

3.4.1 NAT for Inbound Connections

For load balanced connections, the Host Agent performs stateful layer-4 NAT for all connections. As encapsulated packets arrive at the Host Agent, it decapsulates them and then performs a NAT as per the NAT rules configured by Ananta Manager. The NAT rules describe the rewrite rules of type: $(VIP, protocol_v, port_v) \Rightarrow (DIP, protocol_v, port_d)$. In this case, the Host Agent identifies packets that are destined to $(VIP, protocol_v, port_v)$, rewrites the destination address and port to $(DIP, port_d)$ and creates bi-directional flow state that is used by subsequent packets of this connection. When a return packet for this connection is received, it does reverse NAT based on flow state and sends the packet to the source directly through the router, bypassing the Muxes.

3.4.2 Source NAT for Outbound Connections

For outbound connections, the Host Agent does the following. It holds the first packet of a flow in a queue and sends a message to Ananta Manager requesting a VIP and port for this connection. Ananta Manager responds with a VIP and port and the Host Agent NATs all pending connections to different destinations using this VIP and port. Any new connections to different destinations $(remoteaddress, remoteport)$ can also reuse the same port as the TCP five-tuple will still be unique. We call this technique *port reuse*. AM may return multiple ports in response to a single request. The HA uses these ports for any subsequent connections. Any unused ports are returned back after a configurable idle timeout. If the HA keeps getting new connections, these ports are never returned back to AM, however, AM may force HA to release them at any time. Based on our production workload, we have made a number of optimizations to minimize the number of SNAT requests a Host Agent needs to send, including preallocation of ports. These and other optimizations are discussed later in this paper.

3.4.3 DIP Health Monitoring

Ananta is responsible for monitoring the health of DIPs that are behind each VIP endpoint and take unhealthy DIPs out of rotation. DIP health check rules are specified as part of VIP Configuration. On first look, it would seem natural to run health monitoring on the Mux nodes so that health monitoring traffic would take the exact same network path as the actual data traffic. However, it would put additional load on each Mux, could result in a different health state on each mux and would incur additional monitoring load on the DIPs as the number of Muxes can be large. Guided by our principle of offloading to end systems, we chose to implement health monitoring on the Host Agents. A Host Agent monitors the health of local VMs and communicates any changes in health to AM, which then relays these messages to all Muxes in the Mux Pool. Perhaps surprising to some readers, running health monitoring on the host makes it easy to protect monitoring endpoints against unwarranted traffic – an agent in the guest VM learns the host VM's IP address via DHCP and configures a firewall rule to allow monitoring traffic only from the host. Since a VM's host address does not change (we don't do live VM migration), migration of a VIP from one instance of Ananta to another or scaling the number of Muxes does not require reconfiguration inside guest VMs. We believe that these benefits justify this design choice. Furthermore, in a fully managed cloud environment such as ours, out-of-band monitoring can detect network partitions where HA considers a VM healthy but some Muxes are unable to communicate with its DIPs, raise an alert and even take corrective actions.

3.5 Ananta Manager

The Ananta Manager (AM) implements the control plane of Ananta. It exposes an API to configure VIPs for load balancing and SNAT. Based on the VIP Configuration, it configures the Host Agents and Mux Pools and monitors for any changes in DIP health. Ananta

Manager is also responsible for keeping track of health of Muxes and Hosts and taking appropriate actions. AM achieves high availability using the Paxos [14] distributed consensus protocol. Each instance of Ananta runs five *replicas* that are placed to avoid correlated failures. Three replicas need to be available at any given time to make forward progress. The AM uses Paxos to elect a primary, which is responsible for performing all configuration and state management tasks. We now look at some key AM functions in detail.

3.5.1 SNAT Port Management

AM also does port allocation for SNAT (§3.2.3). When an HA makes a new port request on behalf of a DIP, AM allocates a free port for the VIP, replicates the allocation to other AM replicas, creates a stateless VIP map entry mapping the port to the requesting DIP, configures the entry on the Mux Pool and then sends the allocation to the HA. There are two main challenges in serving SNAT requests — latency and availability. Since SNAT request is done on the first packet of a connection, a delay in serving SNAT request directly translates into latency seen by applications. Similarly, AM downtime would result in failure of outbound connections for applications resulting in complete outage for some applications. Ananta employs a number of techniques to reduce latency and increase availability. First, it allocates a contiguous port range instead of allocating one port at a time. By using fixed sized port ranges, we optimize storage and memory requirements on both the AM and the Mux. On the Mux driver, only the start port of a range is configured and by making the port range size a power of 2, we can efficiently map a range of ports to a specific DIP. Second, it preallocates a set of port ranges to DIPs when it first receives a VIP configuration. Third, it tries to do demand prediction and allocates multiple port ranges in a single request. We evaluate the effectiveness of these techniques in §5.

3.6 Tenant Isolation

Ananta is a multi-tenant load balancer and hence tenant isolation is an essential requirement. The goal of tenant isolation is to ensure that the *Quality of Service* (QoS) received by one tenant is independent of other tenants in the system. However, Ananta is an oversubscribed resource and we do not guarantee a minimum bandwidth or latency QoS. As such, we interpret this requirement as follows – the total CPU, memory and bandwidth resources are divided among all tenants based on their weights. The weights are directly proportional to the number of VMs allocated to the tenant. We make another simplifying assumption that traffic for all VIPs is distributed equally among all Muxes, therefore, each Mux can independently implement tenant isolation. We find this assumption to hold true in our environment. If this assumption were to not hold true in the future, a global monitor could dynamically inform each Mux to assign different weights to different tenants. Memory fairness is implemented at Muxes and AM by keeping track of flow state and enforcing limits.

3.6.1 SNAT Fairness

AM is a critical resource as it handles all SNAT port requests. Excessive requests from one tenant should not slow down the SNAT response time of another tenant. AM ensures this by processing requests in a first-come-first-serve (FCFS) order and it ensures that at any given time there is at most one outstanding request from a DIP. If a new request is received while another request from the same DIP is pending, the new request is dropped. This simple mechanism ensures that VIPs get SNAT allocation time in proportion to the number of VMs. Furthermore, there are limits on the number of ports allocated and rate of allocations allowed for any given VM.

3.6.2 Packet Rate Fairness

Mux tries to ensure fairness among VIPs by allocating available bandwidth among all active flows. If a flow attempts to steal more than its fair share of bandwidth, Mux starts to drop its packets with a probability directly proportional to the excess bandwidth it is using. While bandwidth fairness works for TCP flows that are sending large-sized packets, it does not work for short packets spread across flows, or flows that are non-TCP (e.g., UDP) or flows that are malicious (e.g., a DDoS attack). A key characteristic of these flows is that they do not back off in response to packet drops, in fact we sometimes see the exact opposite reaction. Since dropping packets at the Mux is not going to help and increases the chances of overload, our primary approach has been to build a robust detection mechanism for overload due to packet rate. Each Mux keeps track of its *top-talkers* – VIPs with the highest rate of packets. Mux continuously monitors its own network interfaces and once it detects that there is packet drop due to overload, it informs AM about the overload and the top talkers. AM then identifies the topmost top-talker as the victim of overload and withdraws that VIP from all Muxes, thereby creating a *black hole* for the VIP. This ensures that there is minimal collateral damage due to Mux overload. Depending on the policy for the VIP, we then route it through DoS protection services (the details are outside the scope of this paper) and enable it back on Ananta. We evaluate the effectiveness of overload detection and route withdrawal in §5.

3.7 Design Alternatives

3.7.1 DNS-based Scale Out

Ananta uses BGP to achieve scale out among multiple active instances of Mux. A traditional approach to scaling out middlebox functionality is via DNS. Each instance of the middlebox device, e.g., load balancer, is assigned a public IP address. The authoritative DNS server is then used to distribute load among IP addresses of the instances using an algorithm like weighted round-robin. When an instance goes down, the DNS server stops giving out its IP address. This approach has several limitations. First, it is harder to get good distribution of load because it is hard to predict how much load can be generated via a single DNS resolution request. For example, load from large clients such as a *megaproxy* is always sent to a single server. Second, it takes longer to take unhealthy middlebox nodes out of rotation due to DNS caching – many local DNS resolvers and clients violate DNS TTLs. And third, it cannot be used for scale out of stateful middleboxes, such as a NAT.

3.7.2 OpenFlow-based Load Balancing

An alternative to implementing Mux functionality in general-purpose servers is to use OpenFlow-capable switches [16]. However, currently available OpenFlow devices have insufficient support for general-purpose layer-4 load balancing. For example, existing OpenFlow switches only support 2000 to 4000 flows, whereas Mux needs to maintain state for millions of flows. Another key primitive lacking in existing OpenFlow hardware is tenant isolation. Finally, in a pure OpenFlow-based network, we will also need to replace BGP with centralized routing in Ananta Manager, which has certain drawbacks as discussed in §6. Other researchers have also attempted to build load balancers using OpenFlow switches [28], however, these solutions do not yet meet all the requirements, e.g., tenant isolation and scalable source NAT across layer-2 domains.

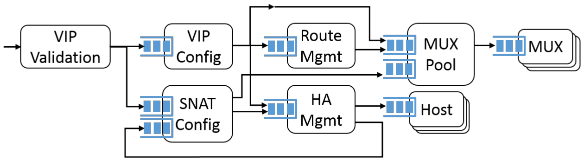


Figure 10: Staged event-driven (SEDA) Ananta Manager. *Ananta manager shares the same threadpool across multiple stages and supports priority event queues to maintain responsiveness of VIP configuration operations under overload.*

4. IMPLEMENTATION

We implemented all three Ananta components from Figure 5. Ananta Manager and Mux are deployed as a tenant of our cloud platform itself. The Host Agent is deployed on every host in our cloud and is updated whenever we update the Ananta tenant. In a typical deployment five replicas of AM manage a single Mux Pool. Most Mux Pools have eight Muxes in them but the number can be based on load.

Ananta Manager: AM performs various time-critical tasks — configuration of VIPs, allocation of ports, coordination of DIP health across Muxes. Therefore, its responsiveness is very critical. To achieve a high degree of concurrency, we implemented AM using a lock-free architecture that is somewhat similar to SEDA [29]. As shown in Figure 10, AM is divided into the following stages — VIP validation, VIP configuration, Route Management, SNAT Management, Host Agent Management and Mux Pool Management. Ananta implementation makes two key enhancements to SEDA. First, in Ananta, multiple stages share the same threadpool. This allows us to limit the total number of threads used by the system. Second, Ananta supports multiple priority queues for each stage. This is useful in maintaining responsiveness during overload conditions. For example, SNAT events take less priority over VIP configuration events. This allows Ananta to finish VIP configuration tasks even when it is under heavy load due to SNAT requests.

Ananta maintains high availability using Paxos [14]. We took an existing implementation of Paxos and added discovery and health monitoring using the SDK of our cloud platform. The SDK notifies AM of addition, removal or migration of any replicas of the Paxos cluster. This allows us to automatically reconfigure the cluster as instances are added or removed dynamically. These features of the platform SDK result in significantly reduced operational overhead for Ananta. The platform also provides a guarantee that no more than one instance of the AM role is brought down for OS or application upgrade. This notion of instance-by-instance update maintains availability during upgrades. Ananta uses Paxos to elect a primary and only the primary does all the work.

Mux: Mux has two main components — a kernel-mode driver and a user-mode BGP [20] speaker. The kernel-mode driver is implemented using the Windows Filtering Platform (WFP) [30] driver model. The driver intercepts packets at the IP layer, encapsulates and then sends them using the built-in forwarding function of the OS. Delegating routing to the OS stack has allowed Mux code to remain simple, especially when adding support for IPv6, as it does not need to deal with IP fragmentation and next-hop selection. The driver scales to multiple cores using receive side scaling (RSS) [22]. Since the Mux driver only encapsulates the packet with a new IP header and leaves the inner IP header and its payload intact, it does not need to recalculate TCP checksum and hence it does not need any sender-side NIC offloads. For IPv4, each Mux can hold 20,000 load balanced endpoints and 1.6 million SNAT ports in its VIP Map with 1GB of memory. Each mux can maintain state for millions of connections and is only limited by available memory on the server. CPU performance characteristics of the Mux are discussed in §5.2.3.

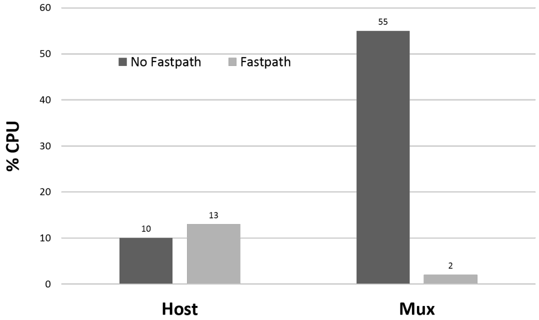


Figure 11: CPU usage at Mux and Hosts with and without Fastpath. *Once Fastpath is turned on, the hosts take over the encapsulation function from Mux. This results in lower CPU at Mux and CPU increase at every host doing encapsulation.*

Host Agent: The Host Agent also has a driver component that runs as an extension of the Windows Hyper-V hypervisor's virtual switch. Being in the hypervisor enables us to support unmodified VMs running different operating systems. For tenants running with native (non-VM) configuration, we run a stripped-down version of the virtual switch inside the host networking stack. This enables us to reuse the same codebase for both native and VM scenarios.

Upgrading Ananta: Upgrading Ananta is a complex process that takes place in three phases in order to maintain backwards-compatibility between various components. First, we update instances of the Ananta Manager, one at a time. During this phase, AM also adapts its persistent state from previous schema version to the new version. Schema rollback is currently not supported. Second, we upgrade the Muxes; and third, the Host Agents.

5. MEASUREMENTS

In this section we first present a few micro-benchmarks to evaluate effectiveness and limitations of some of our design choices and implementation, and then some data from real world deployments.

5.1 Micro-benchmarks

5.1.1 Fastpath

To measure the effectiveness of Fastpath and its impact on host CPU, we conducted the following experiment. We deployed a 20 VM tenant as the server and two 10 VM tenants as clients. All the client VMs create up to ten connections to the server and upload 1MB of data per connection. We recorded the CPU usage at the host nodes and one Mux. The results are shown in Figure 11. We found the host CPU usage to be uniform across all hosts, so we only show median CPU observed at a representative host. As expected, as soon as Fastpath is turned on, no new data transfers happen through the Mux. It only handles the first two packets of any new connection. Once the Mux is out of the way, it also stops being a bottleneck for data transfer and VMs can exchange data at the speed allowed by the underlying network.

5.1.2 Tenant Isolation

Tenant isolation is a fundamental requirement of any multi-tenant service. In the following we present two different experiments

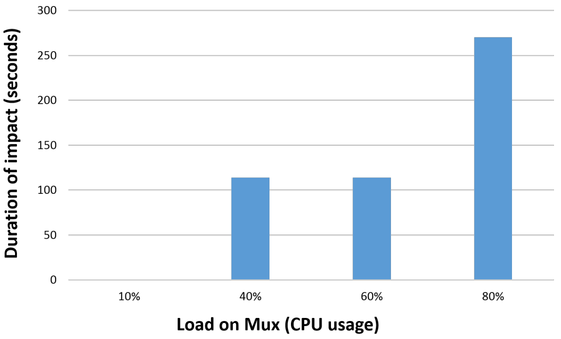


Figure 12: SYN-flood Attack Mitigation. *Duration of impact shows the time Ananta takes to detect and black-hole traffic to the victim VIP on all Muxes.*

that show Ananta's ability to isolate inbound packet and outbound SNAT abuse. For these experiments, we deployed five different tenants, each with ten virtual machines, on Ananta.

SYN-flood Isolation: To measure how quickly Ananta can isolate a VIP under a SYN-flood attack, we ran the following experiment (other packet rate based attacks, such as a UDP-flood, would show similar result.) We load Ananta Muxes with a baseline load and launch a SYN-flood attack using spoofed source IP addresses on one of the VIPs. We then measure if there is any connection loss observed by clients of the other tenants. Figure 12 shows the maximum duration of impact observed over ten trials. As seen in the chart, Mux can detect and isolate an abusive VIP within 120 seconds when it is running under no load, minimum time being 20 seconds. However, under moderate to heavy load it takes longer to detect an attack as it gets harder to distinguish between legitimate and attack traffic. We are working on improving our DoS detection algorithms to overcome this limitation.

SNAT Performance Isolation: SNAT port allocation at Ananta Manager could be a subject of abuse by some tenants. It is possible that a tenant makes a lot of SNAT connections causing impact on other tenants. To measure the effectiveness of per-VM SNAT isolation at AM, we conducted the following experiment. A set of normal use tenants (N) make outbound connections at a steady rate of 150 connections per minute. Whereas, a heavy SNAT user (H) keeps increasing its SNAT requests. We measure the rate of SYN retransmits and the SNAT response time of Ananta at the corresponding HAs. Figure 13 shows the aggregate results over multiple trials. As seen in the figure, the normal tenants' connections keep succeeding at a steady rate without any SYN loss; and its SNAT port requests are satisfied within 55ms. The heavy user, on the other hand, starts to see SYN retransmits because Ananta delays its SNAT requests in favor of N. This shows that Ananta rewards good behavior by providing faster SNAT response time.

5.1.3 SNAT Optimizations

A key design choice Ananta made is to allocate SNAT ports in AM and then replicate the port allocations to all Muxes and other AM replicas to enable scale out and high availability respectively. Since port allocation takes place for the first packet of a connection, it can add significant latency to short connections. Ananta overcomes these limitations by implementing three optimizations as mentioned in §3.5.1. To measure the impact of these optimizations we conducted the following experiment. A client continuously makes outbound TCP connections via SNAT to a remote service and records the connection establishment time. The resulting data is partitioned into buckets of 25ms. The minimum connection establishment time to the remote service (without SNAT) is 75ms. Figure 14 shows connection establishment times for the following two optimizations when there is no other load on the system.

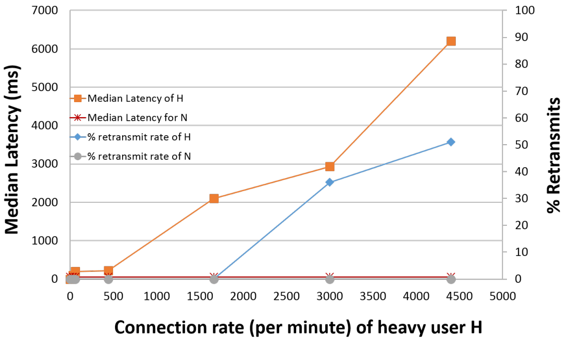


Figure 13: Impact of heavy SNAT user H on a normal user N. *Heavy user H sees higher latency and higher SYN retransmits. Normal user N's performance remains unchanged.*

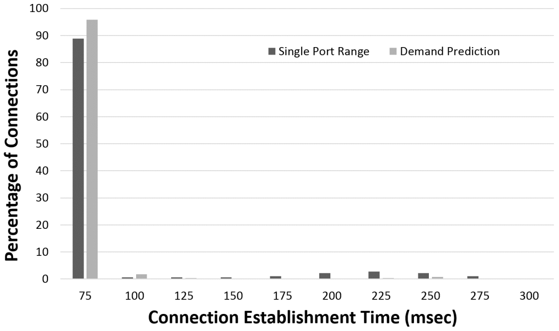


Figure 14: Connection establishment time experienced by outbound connections with and without port demand prediction.

Single Port Range: In response to a port request, AM allocates eight contiguous ports instead of a single port and returns them to the requesting HA. HA uses these ports for any pending and subsequent connections. The HA keeps any unused ports until a preconfigured timeout before returning them to AM. By doing this optimization, only one in every eight outbound connections ever results in a request to AM. This is evident from Figure 14 — 88% connections succeed in the minimum possible time of 75ms. The remaining 12% connections take longer due to the round-trip to AM. Without the port range optimization every new connection request that cannot be satisfied using existing already allocated ports will make a round-trip to AM.

Demand Prediction: When this optimization is turned on, AM attempts to predict port demand of a DIP based on its recent history. If a DIP requests new ports within a specified interval from its previous requests, AM allocates and returns multiple eight-port port ranges instead of just one. As shown in Figure 14, with this optimization 96% of connections are satisfied locally and don't need to make a round-trip to AM. Furthermore, since AM handles fewer requests, its response time is also better than allocating a single port range for each request.

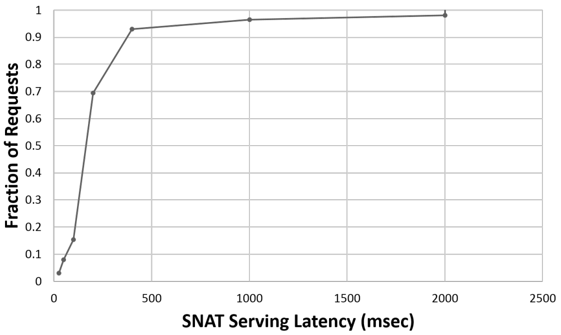


Figure 15: CDF of SNAT response latency for the 1% requests handled by Ananta Manager.

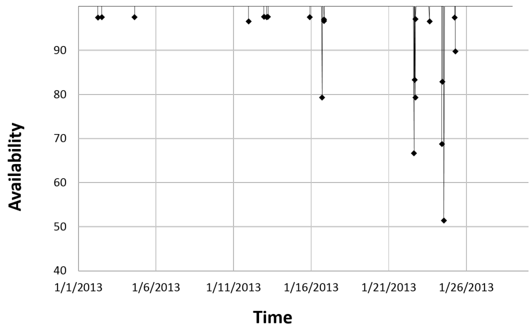


Figure 16: Availability of test tenants in seven different data centers over one month.

5.2 Real World Data

Several instances of Ananta have been deployed in a large public cloud with a combined capacity exceeding 1Tbps. It has been serving Internet and intra-data center traffic needs of very diverse set of tenants, including blob, table and queue storage services for over two years. Here we look at some data from real world.

5.2.1 SNAT Response Latency

Based on production data, Ananta serves 99% of the SNAT requests locally by leveraging port reuse and SNAT preallocation as described above. The remaining requests for tenants initiating a lot of outbound requests to a few remote destinations require SNAT port allocation to happen at the AM. Figure 15 shows the distribution of latency incurred for SNAT port allocation over a 24 hour window in a production data center for the requests that go to AM. 10% of the responses are within 50ms, 70% within 200ms and 99% within 2 seconds. This implies that in the worst case, one in every 10000 connections suffers a SYN latency of 2 seconds or higher; however, very few scenarios require that number of connections to the same destination.

5.2.2 Availability

As part of ongoing monitoring for our cloud platform, we have multiple test tenants deployed in each data center. A monitoring service connects to the VIP of every test tenant from multiple geographic locations and fetches a web page once every five minutes.

Figure 16 shows average availability of test tenants in seven different data centers. If the availability was less than 100% for any five minute interval, it makes up a point in the above graph. All the other intervals had 100% availability. The average availability over this period across all test tenants was 99.95%, with a minimum of 99.92% for one tenant and greater than 99.99% for two tenants. Five of the low availability conditions between Jan 21 and Jan 26 happened due to Mux overload. The Mux overload events were primarily caused by SYN-flood attacks on some tenants that are not protected by our DoS protection service. Two availability drops were due to wide-area network issues while the rest were false positives due to update of the test tenants.

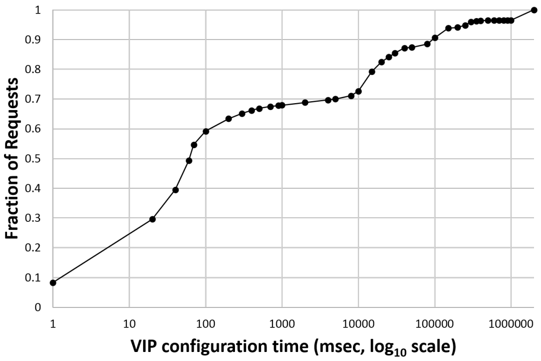


Figure 17: Distribution of VIP configuration time over a 24-hr period.

5.2.3 Scale

For control plane, Ananta ensures that VIP configuration tasks are not blocked behind other tasks. In a public cloud environment, VIP configuration tasks happen at a rapid rate as customers add, delete or scale their services. Therefore, VIP configuration time is an important metric. Since Ananta needs to configure HAs and Muxes during each VIP configuration change, its programming time could be delayed due to slow HAs or Muxes. Figure 17 shows the distribution of time taken by seven instances Ananta to complete VIP configuration tasks over a 24-hr period. The median configuration time was 75ms, while the maximum was 200seconds. These time vary based on the size of the tenant and the current health of Muxes. These times fit within our API SLA for VIP configuration tasks.

For data plane scale, Ananta relies on ECMP at the routers to spread load across Muxes and RSS at the NIC to spread load across multiple CPU cores. This design implies that the total upload throughput achieved by a single flow is limited by what the Mux can achieve using a single CPU core. On our current production hardware, the Mux can achieve 800Mbps throughput and 220Kpps using a single x64, 2.4GHz core. However, Muxes can achieve much higher aggregate throughput across multiple flows. In our production deployment, we have been able to achieve more than 100Gbps sustained upload throughput for a single VIP. Figure 18 shows bandwidth and CPU usage seen over a typical 24-hr period by 14 Muxes deployed in one instance of Ananta. Here each Mux is running on a 12-core 2.4Ghz intel$^{®}$ Xeon CPU. This instance of Ananta is serving 12 VIPs for blob and table storage service. As seen in the figure, ECMP is load balancing flows quite evenly across Muxes. Each Mux is able to achieve 2.4Gbps throughput (for a total of 33.6Gbps) using about 25% of its total processing power.

6. OPERATIONAL EXPERIENCE

We have been offering layer-4 load balancing as part of our cloud platform for the last three years. The inability of hardware load balancers to deal with DoS attacks, network elasticity needs, the grow-

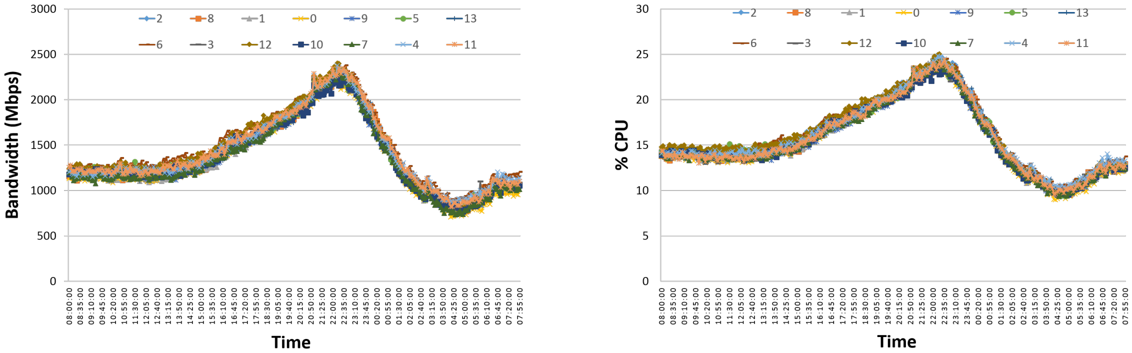


Figure 18: Bandwidth and CPU usage over a 24-hr period for 14 Muxes in one instance of Ananta.

ing bandwidth requirements and increasing price pressure made us finally give up on hardware load balancers. Having our own software-based load balancer has not been without any issues either, however, we have been able to troubleshoot and fix those issues much faster – in days instead of weeks or months. Here we look at some of those issues.

Ananta Manager uses Paxos to elect a primary replica that does all the work. During the initial stages of Ananta, we assumed that there can never be two replicas claiming to be primary at the same time. However, we encountered issues where this assumption was not true. This happens due to old hard disks where the disk controller would freeze for two minutes or longer on the primary replica. During this freeze, the secondary replicas would conclude that the primary is down and elect a new primary. But once the disk controller on the old primary becomes responsive again, it continues to do work assuming it is still the primary for a short period of time. This resulted in customers facing outages because all the host agents would continue to send health reports to the old primary while Muxes would only accept commands from the new primary. We fixed this issue by having the primary perform a Paxos write transaction whenever a Mux rejected its commands. This change resulted in the old primary detecting its stale status as soon as it would try to take any action.

We made an explicit design choice to use encapsulation to deliver packets between the Muxes and hosts so that Muxes could load balance across layer-2 domains. This lowers the effective MTU available for transport layer payload. In order to avoid fragmentation of packets at the Mux, which incurs high CPU overhead, Ananta Host Agents adjust the MSS value exchanged as part of TCP connection establishment to a lower value – 1440 from 1460 for IPv4. This technique worked perfectly fine for almost two years and then we suddenly started seeing connection errors because clients were sending full-sized packets (1460 byte payload) with IP *Don't Fragment* (DF) bit set. After encapsulation at the Muxes the ethernet frame size (1520) exceeded network MTU of 1500, resulting in packet drops. This started to happen because of two external bugs. A specific brand of home routers has a bug in that it always overwrites the TCP MSS value to be 1460. Therefore, clients never see the adjusted MSS. Normally this is fine because clients are supposed to retry lost full-sized TCP segments by sending smaller-sized segments. However, there was a bug in the TCP implementation of a popular mobile operating system in that TCP retries used the same full-sized segments. Since we control our entire network, we increased the MTU on our network to a higher value so that it can accommodate encapsulated packets without fragmentation.

One design decision that has been a subject of many debates is collocation of data plane and its control via BGP on Mux. An alternative design would be to host BGP in a separate service and control it from AM. Collocation works better for hard failures as routers quickly take an unhealthy Mux out of rotation due to BGP timeout. However, when incoming packet rate exceeds the processing capacity of a Mux, the BGP traffic gets affected along with data traffic. This causes the Mux to lose BGP connection to the router and that traffic shifts to another Mux, which in turn gets overloaded and goes down, and so on. This cascading failure can bring down all the Muxes in a cluster. There are multiple ways to address this limitation. One solution is to use two different network interfaces – one for control plane and the other for data plane. Another solution is to rate limit data traffic at the router so that there is always sufficient headroom for control traffic. The same mechanisms will be required to separate control traffic from data traffic even if BGP was hosted in a separate service. Overall, we found that collocation leads to a simpler design in this case.

One of the issues with using hardware load balancers was that we had to set an aggressive idle connection timeout (60 seconds), otherwise legitimate connections were getting dropped under heavy load, e.g., under connection state-based attacks. In the initial versions of Ananta, we kept the same aggressive idle timeout. However, with the increase in the number of low-power, battery-operated mobile devices, there was a need to keep connections open for a long time even where there is no active traffic. For example, a phone notification service needs to push data immediately to such devices without requiring them to send frequent *keep-alive* messages. We were able to increase idle timeout in Ananta because it keeps the NAT state on the hosts and, under overload, the Muxes can continue forwarding traffic using the VIP map (§3.3) without creating new per-connection state. This makes Ananta significantly less vulnerable to connection state-based attacks.

7. RELATED WORK

To the best of our knowledge, Ananta takes a new approach to building layer-4 distributed load balancer and NAT that meets the requirements of a multi-tenant cloud environment. In this section we compare it to the state-of-the-art in industry and research. We recognize that many existing solutions provide more than layer-4 load balancing and their design choices may have been influenced by those additional features. Ananta's modular approach can be used to build application-layer functionality, such as layer-7 load balancing, on top of the base layer-4 functionality.

Traditional hardware load balancers (e.g., [9, 1]) have a scale up design instead of a scale out design. Furthermore, they provide $1 + 1$ redundancy, which can leave the system with no redundancy during times of repair and upgrade. Cloud environments need $N+1$ redundancy to meet the high uptime requirements. Another key challenge with hardware appliances is their inability to scale up or down with demand.

Many vendors now provide load balancing software that can run in virtual machines on general purpose server hardware [17, 15]. Some of these solutions are based on the open source HA Proxy software [12]. These virtual appliances have two limitations that made them unsuitable for our cloud environment. First, they provide $1 + 1$ redundancy, similar to hardware load balancers. Operators work around this limitation by migrating IP addresses from the failed VM to a spare, which forces them to deploy multiple load balancers in the same layer-2 domain leading to fragmentation of capacity. Second, these appliances cannot scale a single VIP beyond the capacity of a single VM. With this design, any service that needs to scale beyond the capacity of a single device uses multiple VIPs, which has several drawbacks as discussed in §3.7.

Embrane [8] promises the benefits of using software, including scale out. Ananta differs from Embrane in that it offloads signficant functionality to the Host Agent and leverages existing routers for scale out. Egi et al [7] show that high performance routers can be built using commodity hardware. More recent efforts, such as RouteBricks [6], have shown that a commodity server can easily achieve 12.8Gbps for minimum-sized packets on commodity hardware. By carefully partitioning the load balancing workload, we have reduced the in-network processing to be similar to routing. Therefore, these results for routing support our approach of implementing data plane on commodity hardware.

ETTM [26] is similar to Ananta in many aspects, e.g., every end-host participates in packet processing in both designs. However, a key difference is that Ananta implements in-network routing in dedicated commodity servers, which enables it to overcome the limited packet modification options in current hardware.

8. CONCLUSION

Ananta is a distributed layer-4 load balancer and NAT specifically designed to meet the scale, reliability, tenant isolation and operational requirements of multi-tenant cloud environments. While its design was heavily influenced by the Windows Azure public cloud, many design decisions may apply to building other middleboxes that require similarly scalable packet processing at low cost, e.g., intrusion detection systems or virtual private network (VPN).

We started by reexamining the requirements of a large-scale cloud and concluded that we needed a solution that scales with the size of the network while maintaining low cost and operational overhead. These requirements led us to build a scale-out data plane by leveraging existing routing primitives and offloading some heavy packet processing tasks to the host. We coordinate state across the distributed data plane using a highly available control plane. This design enables direct server return and scalable NAT across layer-2 domains. Furthermore, within the data center, the load balancer gets out of the way of normal traffic flow, thereby enabling unfettered network bandwidth between services. Ananta delivers tenant isolation by throttling and isolating heavy users.

Over a 100 instances of Ananta have been deployed in the Windows Azure public cloud, serving over 100,000 VIPs. Building our own software solution gives us control over a very important vantage point in our network as it sees most incoming and outgoing traffic and is key to maintaining high availability for our tenants.

Acknowledgements

The many comments from our shepherd John Heidemann and anonymous reviewers greatly improved the final version of this paper. We would like to thank additional contributors to Ananta: Yue Zuo, Nancy Williams, Somesh Chaturmohta, Narayanan Sethuraman, Jayendran Srinivasan, Vladimir Ivanov, Nisheeth Srivastava, Naveen Prabhat, Shangguang Wu, Thiruvengadam Venketesan, Luis Irun-Briz, Narayan Annamalai, Lin Zhong, Greg Lapinski, Ankur Agiwal and Deepali Bhardwaj.

We would also like to thank the following members of the Autopilot team with whom we shared principles and insights gained as they built a production load balancer to support services like bing.com: Saby Mahajan, Chao-Chih Chen, Pradeep Mani, Chao Zhang, Arun Navasivasakthivelsamy, Shikhar Suri and Long Zhang.

Finally, we would like to thank Yousef Khalidi, Reza Baghai, G S Rana, Mike Neil, Satya Nadella, and the rest of the Windows Azure team for their support.

9. REFERENCES

[1] A10 Networks AX Series. http://www.a10networks.com.

[2] Aryaka WAN Optimization. http://www.aryaka.com.

[3] Amazon Web Services. http://aws.amazon.com.

[4] Microsoft Windows Azure. http://www.windowsazure.com.

[5] T. Benson, A. Akella, A. Shaikh, and S. Sahu. CloudNaaS: A Cloud Networking Platform for Enterprise Applications. In *Symposium on Cloud Computing*, 2011.

[6] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. RouteBricks: exploiting parallelism to scale software routers. In *SOSP*, 2009.

[7] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, and L. Mathy. Towards high performance virtual routers on commodity hardware. In *CoNEXT*, 2008.

[8] Embrane. http://www.embrane.com.

[9] F5 BIG-IP. http://www.f5.com.

[10] Google Cloud Platform. http://cloud.google.com.

[11] A. Greenberg et al. VL2: A scalable and flexible data center network. In *SIGCOMM*, 2009.

[12] HA Proxy Load Balancer. http://haproxy.1wt.eu.

[13] A. Heffernan. RFC 2385: Protection of BGP Sessions via the TCP MD5 Signature Option, 1998.

[14] L. Lamport. The Part-Time Parliament. *ACM TOCS*, 16(2):133–169, May 1998.

[15] LoadBalancer.org Virtual Appliance. http://www.load-balancer.org.

[16] N. Mckeown, T. Anderson, H. Balakrishnan, G. M. Parulkar, L. L. Peterson, J. Rexford, S. Shenker, and J. S. Turner. OpenFlow: Enabling Innovation in Campus Networks. In *SIGCOMM*, 2008.

[17] NetScalar VPX Virtual Appliance. http://www.citrix.com.

[18] C. Perkins. RFC 2003: IP Encapsulation within IP, 1996.

[19] Rackspace. http://www.rackspace.com.

[20] Y. Rekhter, T. Li, and S. Hares. RFC 4271: A Border Gateway Protocol 4 (BGP-4), 2006.

[21] Riverbed Virtual Steelhead. http://www.riverbed.com.

[22] Receive Side Scaling. http://msdn.microsoft.com.

[23] V. Sekar, S. Ratnasamy, M. K. Reiter, N. Egi, and G. Shi. The Middlebox Manifesto: Enabling Innovation in Middlebox Deployment. In *HotNets*, 2011.

[24] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service. In *SIGCOMM*, 2012.

[25] D. Thaler and C. Hopps. RFC 2991: Multipath Issues in Unicast and Multicast Next-Hop Selection, 2000.

[26] H. Uppal, V. Brajkovic, D. Brandon, T. Anderson, and A. Krishnamurthy. ETTM: A Scalable Fault Tolerant Network Manager. In *NSDI*, 2011.

[27] Vyatta Software Middlebox. http://www.vyatta.com.

[28] R. Wang, D. Butnariu, and J. Rexford. OpenFlow-Based Server Load Balancing GoneWild. In *Hot-ICE*, 2011.

[29] M. Welsh, D. Culler, and E. Brewer. SEDA: An Architecture for Well-Conditioned, Scalable Internet Services. In *SOSP*, 2001.

[30] Windows Filtering Platform. http://msdn.microsoft.com.

[31] ZScalar Cloud Security. http://www.zscalar.com.