PacketCloud: Hosting In-network Services in a Cloud-like Environment

Yang Chen, Ang Li, Xiaowei Yang
Department of Computer Science, Duke University
{ychen, angl, xwy}@cs.duke.edu
Duke CS-TR-2011-10

1. INTRODUCTION

There are two general types of network services in today's Internet. The first type of services, so-called the *end-to-end* services, requires the Internet only handle the end-to-end data delivery. Any additional computation and storage tasks should be handled by end hosts. We are most familiar with such services, as most of our daily Internet applications, such as WWW, email, and VoIP, all belong to this category.

The second type of services, which we refer to as the *in-network* services, are not as well-known. Different from the end-to-end ones, the in-network services serve the users' traffic inside the network, and they are usually not the final destinations of the traffic. In-network services are in a unique position to help the network better handle user's traffic, because they have access to the network layer information, and can modify packets while they are still on-the-fly. For instance, an in-network service can filter out DoS traffic near their sources to save precious core bandwidth [16], also, in-network packet caches can be hosted for eliminating redundant traffic sent over a link [2]. New forwarding schemes such as source routing and multicast can also be supported with in-network services [20, 22]. Recently, the wider usage of mobile devices and the increasing demand for energy-hungry applications rise the requirement of an infrastructure for hosting in-network services, thus the some computational intensive tasks such as payload encryption/decryption or transcoding can be offloaded to the network. Such remote execution [7] can save the limited battery capacity significantly.

Although in-network services are useful, it is challenging to deploy them in today's Internet. First, the core of the Internet infrastructure is well-known to be ossified [25]: the hardware routers have very limited programmability, and the installation of new middle-boxes is costly and inconvenient. Much work has proposed new programmable routers using either commodity PCs [9, 17, 14] or programmable hardwares [3, 18]. However, replacing the existing legacy routers with the new ones can be costly for the ISPs because they not only need to buy the new facilities, but also need to consider how to get refund from retired ones. The ISPs may also raise reliability concerns as the new technologies are still maturing. Furthermore, even if the new programmable routers are ready for production use, the ISPs may not have enough incentives to deploy them, because currently there is no viable economic model for the in-network services.

In this paper, we seek to make the deployment of in-network services easier, by proposing a new hosting platform for the services called PacketCloud. PacketCloud, inspired by the public cloud computing platforms [1], has two prominent features. First, it uses a small packet processing cluster, so-called a PacketCloud site, to bring programmability to a legacy router. Each PacketCloud site is co-located with a legacy router, and all in-network services are hosted by the site instead of the router. The legacy router is only responsible for forwarding traffic to its co-located PacketCloud site for processing. In other words, PacketCloud is backward compatible with legacy routers, and can be incrementally deployed without harming the existing infrastructure. We show several use cases in Section 2.2. Comparing with the data centers of public cloud which are located in limited places, the PacketCloud sites are much easier to deploy thus the data delivery detour can be significantly shorten.

Second, PacketCloud has an economic model built-in to motivate ISPs to deploy the platform. Besides improving the user experience by deploying more services to attract mobile users, or improving the infrastructure throughput by reducing the duplicate traffic, the ISPs can open some revenue resources by leasing them to third-party network service providers. Similar to the pay-as-you-go charging model of the current public cloud providers, an ISP can also charge these third-party service providers by how much resource the services have consumed. This approach can remedy resource wasting due to over-provisioning, while making economic profits for ISPs. Our preliminary cost analysis in § 3 shows the cost and potential revenue of operating a PacketCloud site.

A few key challenges arise when we design the PacketCloud platform. First, how to design a flexible yet scalable packet processing "cloud"? Each PacketCloud site needs to support multiple services with very different packet processing rules, and also must scale them up to handle traffic peaks. Second, how to prevent malicious services from abusing the resources in the PacketCloud sites? Third, how do both the source and the destination of a packet express which services to be applied to the packet? In the next section, we describe our proposed system design that tries to tackle these challenges.

2. DESIGN

Before discussing the detailed design, we first highlight

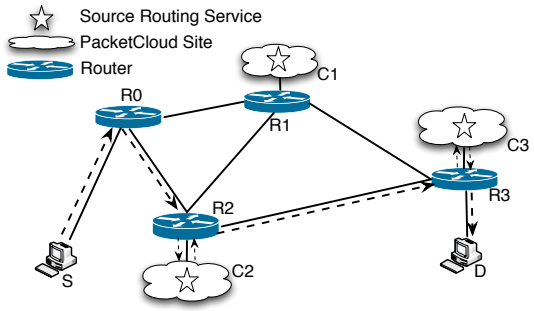

Figure 1: Overall workflow of PacketCloud

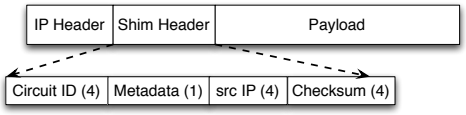

Figure 2: Packet Format for Source-controlled Services

the main design goals of PacketCloud:

1. **Compatible with legacy routers:** We do not want to replace the existing forwarding plane with a new programmable one. Instead, we want to incrementally introduce computation capability and programmability to the existing routers through their co-located PacketCloud sites.

2. **Flexible and scalable packet processing:** Different in-network services have different packet processing rules. On the other hand, some services need to process Gbps of traffic with low forwarding latency. We strive to make the PacketCloud sites capable to support flexible packet processing rules, while at the same time can scale up to handle large amount of traffic. In PacketCloud, we adopt the Platform-as-a-Service paradigm [4] to make developers focus on packet processing logic and to enhance security.

3. **Flexible service selection for end users:** We believe both the source and the destination of a packet have a say in what services should process the packet. The source should be able to explicitly request a service for its packet, while the destination should also be able to control how it receives the packets. We therefore design the PacketCloud platform to support the both cases with a flexible service selection model.

4. **Strong economic incentives for ISPs:** We hope to provide strong economic incentives for ISPs to deploy the PacketCloud sites. Therefore, billing support is taken into consideration from day one.

2.1 Overview

The PacketCloud platform involves three roles: the Internet service provider (ISP), the PacketCloud service provider (PSP), and the user of the PacketCloud services. ISPs are responsible to deploy and manage the PacketCloud sites co-located with their routers. PSPs deploy their in-network services inside the chosen PacketCloud sites. Users are the end consumers of the deployed PacketCloud services. Their raw traffic will be served by the requested PacketCloud services on the fly. Note that the three roles are defined from a technical point of view. In practice, one entity might play multiple roles. For instance, an ISP can be a PSP and deploy services in its own PacketCloud sites. A third-party organization can also be a PSP and book some resources of selected PacketCloud sites for hosting a service.

In Fig. 2, we use a simple loose source routing example to illustrate how PacketCloud works. In this example, an ISP's network contains four routers, and three of them have PacketCloud sites deployed. Note that the PacketCloud design does not require a site to be deployed with every router in the network. Similar to Tor [**?**], PacketCloud utilizes a circuit-level end-to-end abstraction. A forwarding circuit will be created between the sender and the receiver, going through one or multiple PacketCloud sites selected according to the interaction between the packet sender and the corresponding PSP. The PSP will notify every on-path PacketCloud site for the information of a new circuit. For every data delivery hop, a four bytes circuit label will be assigned. Note that the circuit labels are hop-specific: one circuit has a different circuit label on each hop, thus this label can be chosen to be locally unique. For each on-circuit PacketCloud site, a unique mapping between the circuit labels of the incoming and outgoing hop will be stored locally, this information can orient the packet to be delivered to next hop (either a downstream PacketCloud site or the final destination) using label switching with a MPLS-like fashion.

In source-controlled services, a PacketCloud-enabled packet is backward compatible with a normal IP packet, and can be forwarded by any legacy router, regardless of whether there exists a co-located PacketCloud site. The sender should insert one 11 bytes shim header between the IP header and the payload of an outgoing packet, as shown in Fig. 1. The shim header contains a four bytes label filed, a one byte metadata field, a four bytes source IP field, and a four bytes checksum field. The label field is used for identifying the corresponding circuit, and the metadata field is used parameter setting and state sharing between different presences (e.g., an encryption service requires an encrypting node and a decrypting node). One PacketCloud site can obtain the next hop of the circuit (circuit label for next hop, and the IP address of next destination) by looking at these two fields. The embedding of the source IP address is used for the receiver to send an ACK packet or even create a feedback tunnel to the sender. Therefore, a service is able to offer flow control and reliable data delivery function for the end users, i.e., by providing specialized socket API which is compatible with PacketCloud architecture. The checksum field is used to verify if the packet is a legal PacketCloud-enabled packet.

To deploy a source routing service inside the ISP, the source routing service provider, a PSP, first develops a small piece of source routing program as follows:

```
In_CirID = pkt.shim.read();
[Out_CirID, nextHop] = fetch(In_CirID);
pkt.shim.update(Out_CirID);
if (nextHop != NULL) forwardTo(nextHop);
else drop();
```

The routine is written in a high-level language similar to the one defined in [20] with various packet processing APIs (§ 2.5). It first reads the latest hop circuit label from a PacketCloud-specific shim header. This label can be used as a key for looking at the local storage, in order to obtain the corresponding circuit label and destination IP address for the next hop. The program calls the `pkt.shim.update()` API for preparing a new shim header for next hop, by updating the circuit label field as the value of the next hop. Afterwards, the program calls the `forwardTo()` API to send the packet to the new destination `nextHop`. This API will update the destination IP filed of the IP header thus the packet can be delivered further through ordinary IP forwarding.

The source routing service provider deploys the service on the ISP by choosing all three PacketCloud sites it wants the service to be running on. The site public IPs ($C1$, $C2$, and $C3$) to the potential users through systems such as DNS. Suppose a source S wants to use the source routing service to send a packet to the destination D through the route $S \rightarrow C2 \rightarrow C3 \rightarrow D$ (Fig. 2). After coordination with the PSP, a circuit will be formed, which includes three hops. The first hop circuit label will be involved in the shim header. Next, it sets the destination IP address of the IP header as $C2$, which is the first PacketCloud site on its route. Ordinary IP forwarding will deliver the packet to $C2$, where the packet is processed using the above source routing program, and further forwarded to $C3$. Eventually, the packet will arrive at D through the indicated source route. Note that this is not the only way how an end user chooses a service for its packets. § 2.4 discusses other ways for service selection.

2.2 Service Selection

PacketCloud supports three forms of service selection for end users. In the simplest source-controlled form, a sender explicitly requests its packets to be served by a service deployed in one or multiple particular PacketCloud sites. This form is suitable for cases such as middlebox traversal and source routing. The circuit creation is completed by the interaction between the sender and the service provider which operates a lookup service. Therefore, service provider has the flexibility to choose the suitable PacketCloud site to serve the users, by considering factors such as latency, bandwidth, traffic engineering, etc. The sender sends the packet to the public IP address of the first PacketCloud site on the forwarding circuit, which will process the packet and forward it further.

In the second form, the sender can request for a service, but does not want to specify (or know how to specify) the PacketCloud site(s) for the service. The form is suitable for network-wide services, such as a congestion-aware forwarding service, where the user does not know or care which PacketCloud sites have processed its packets. In this case, the sender also needs to insert a simplified shim header (3 Bytes Service ID, 1 Bytes metadata, and 3 Bytes checksum) into every outgoing packet to indicate the requested service. On the other hand, it still sets the destination addresses of outgoing packets to be the real destination, instead of any PacketCloud site. We also require the sender to set a special flag (e.g., a special Type-of-Service value) in the IP headers so that a legacy router can distinguish the PacketCloud-enabled packets from normal packets and forward them to its co-located PacketCloud site. Existing techniques such as Policy-Based Routing [6] can be used to recognize the flag.

In the final form, a destination can also implicitly request services for all packets destined to itself. This form is important to implement destination-controlled services such as packet filters. Similar to [25], we require the owner of a destination prefix to sign up with the ISP with proof that it actually owns the prefix. One candidate of the proof can be the signed DNSSEC records of the reverse DNS zone corresponding to the prefix [15]. Upon validating the proof, the ISP then configures its routers to forward the traffic destined to the destination prefix to the co-located PacketCloud sites. The sites will then process the traffic according to the requested service.

Note that the first or the second form can be used simultaneously with the third form, in which case both the sender and the receiver want the packet to be processed by different services. In this case, we always prioritize the receiver's service. For instance, if the receiver's service decides to drop the packet, it will not be processed by the sender's service.

2.3 PacketCloud Site

A PacketCloud site includes four main components: a load balancer, a computation cluster formed by tens of computation nodes, a forwarding information store (FIS), and the internal network. As the only interface between the internal servers and the outside world, the front-end load balancer is responsible to split the incoming traffic to the commodity machines in the computation cluster. A public IP will be assign to this load balancer, thus the co-located router can directly forward all related PacketCloud-enabled packets to it. This IP address will be the only identifier of this site during the packet delivery, thus the users and PSPs do not have to concern about the internal modifications, for example, adding or removing some nodes for the computation cluster. For the first and second service form, the incoming packets will be split after going through this load balancer, which handles a circuit-level load balancing among the computation nodes. Thus the packets belong to the same circuit are forwarded to the same computation node. This is is im-

portant for maintaining the circuit-dependent state. Similar to Flowstream [13], we propose to utilize an Openflow switch for implementing the load balancer, as the interface between the computation nodes and the network, by extracting the latest hop circuit label from the shim header of the packet. A hash function can be chosen for matching every circuit label with the corresponding computation node. For the second and the third service form, since there is no circuit concept, the load balancer can also handle flow-level load balancing in similar way.

The computation cluster is where the actual packet processing takes place, it consists of tens of computation nodes. We adopt the Platform-as-a-Service paradigm [4] for implementing this cluster. A ready to use execution environment will be offered to PSPs.They just need to focus on preparing their service code and uploading the latest version to a central repository. When one node receives a packet to be processed by a service, its runtime environment loads the service program from the central repository, and runs it over the packet. During the execution, the runtime also records the resource usage (CPU cycles, storage space, traffic amount) of the service for billing purpose (§ 2.6). Note that there is no affinity relationship between a service and a computation node, which means any machine is able to run any service deployed in this site. We adopt such a design because it simplifies traffic splitting at the load balancer and it is very easy to scale. Every computation node caches the recent services to amortize the loading overhead. Different from the typical data centers, the size of our computation cluster is very small thus much easier for the deployment. In addition, there is no internal networking overhead among the nodes of the computation cluster.

To make PacketCloud services more flexible, a PacketCloud site also provides a forwarding information store (FIS) to keep service-specific forwarding information. Inspired by the forwarding information bases (FIBs) inside legacy routers, the store keeps the information needed by each service to process packets, and can be updated through an online interface. The FIS is essentially a distributed database formed by several machines. The FIS can store information with different structures, including simple key-value pairs and longest match prefix tables. In the source routing example shown in Section 2.1, one FIS item uses the circuit label of an incoming hop as a key, while storing the corresponding circuit label and the destination IP address of outgoing hop as the value. Each service can retrieve information from the FIS through API calls (§ 2.5). FIS enables a PSP to change the behavior of its service on-the-fly, and can be very useful if the service also has a "control plane", i.e., a higher-layer system that manages the packet processing behaviors (§ 2.2). Further, the ISP itself can also store some network information in the FIS to help network-aware services. For instance, the ISP may publish the congestion status between its sites in the FIS, and an adaptive forwarding service can automatically try to bypass the congested areas. To reduce launching latency, each computation node also caches the recently used FIS items locally.

Category	Description	Examples
Packet I/O	R/W packet content	`read(offset, len)`
Shim Info	Extract the shim header	`shim.read()`
	Update the shim header	`shim.update()`
FIS lookup	Get from FIS	`fetch(key)`, `lookup(table, prefix)`
Caching	Store in local cache	`cache(key, buf)`
Building blocks	Common functions	`aes(buf)`,`compress(buf)`
Logging	Log information	`log(text)`
Delivery	Deliver the packets	`forwardTo(label)`
Drop	Drop the unwanted packets	`drop()`

Table 1: A list of PacketCloud API categories

The internal network of a site is divided into two parts. One is the data network through which the packets are forwarded; the other is the control network that connects the computation machines with the forwarding information store and with the service repository. The two networks are completely separated to minimize interference.

2.4 Packet Processing API

In PacketCloud, each service routine runs in a sandbox environment, and only uses the APIs provided by the platform to manipulate the packets. This make the services providers simple for preparing the service code, while ensuring the security. In this way, an ISP has full control over what information each service can utilize, and how it may modify the packets.

The basic PacketCloud APIs can be categorized into several types, as shown in Table 1. There are a few things to be noted. First, with the packet I/O APIs, a service is not allowed to modify the IP header or the shim header freely. It is only able to update some selective fields, e.g., the destination IP address field for next hop packet delivery, the circuit label for performing label switching, or the metadata field for state sharing between different presences. Therefore, a malicious third-party service cannot spoof the source address or fake a circuit label. Second, local caching is a best-effort service. This is because a service's routine may run on multiple physical machines, and it can incur high overhead to synchronize the caches of different machines.

2.5 Billing

PacketCloud adopts the same pay-as-you-go principle as the current public cloud platforms for charging the third-party services. In particular, each service is charged for the amount of resource it consumes while processing the packets: CPU cycles, the amount of I/Os, bandwidth, etc.

Such billing scheme has three advantages. First, new PSPs entering the market can start with very low cost, as the amount of traffic requesting for its service is low at the beginning. This has the potential to boost innovation, as small startups can deploy their services in large-scale without much upfront investment [4]. Second, it makes the attacks that aim to abuse the infrastructure costly to launch. For instance,

if an attacker wants to congest a link by deliberately creating a forwarding loop, he will then be charged for high bandwidth usage fees, which make the attack economically unattractive. Finally, we charge inter-domain traffic more money than local traffic. This provides strong economic incentive for third-party PSPs to avoid producing unnecessary inter-domain traffic during the circuit creation phase, which is economically attractive to the ISPs.

In PacketCloud, we do not constrain how a PSP may charge the users of its service. Some services may be provided free of charge. A PSP can also set up its own billing and authentication platforms. It may even choose to outsource the billing responsibility to the ISPs. PacketCloud provides the logging API to help account the traffic processed by a service.

2.6 Use Cases

In this subsection we describe several use cases of PacketCloud. Basically, we group them into two categories, i.e., mobile services and general services. We pick up mobile services as a special category due to the rapid development of the usage of mobile devices. For each case, we first describe why it is suitable for PacketCloud deployment, and then outline the service implementation.

2.6.1 Mobile Services

In-Network Mobility Support With the emergence of Internet-capable mobile devices such as smartphones and tablet PCs, the support of roaming across different networks has become more and more important. The traditional mobile IP schemes [8, 19] require the user to maintain a *home agent* as a permanent anchor of a mobile device, and the agent needs to tunnel all traffic destined to the device to the network where the device currently resides. This can cause forwarding detour as all traffic must first reach the home agent before being redirected.

With PacketCloud, we are able to implement a more efficient in-network redirection service for mobile IP. Suppose a customer wants to enable mobility support for a network prefix p. It can deploy a destination-controlled PacketCloud service with prefix p to a set of sites that together cover all traffic destined to p. The service simply tunnels all traffic it receives to the current IP address of the mobile router [8], which then decapsulates the packets and forwards them to devices inside network p. The most recent address of the mobile router is stored in the service's FIS, and is updated by the mobile router whenever the network and the router have moved. The scheme can reduce the forwarding detour as the packets can be redirected inside the core of the network, instead of at the edge.

On-path offloading of computationally intensive tasks The mobile clients has limited battery life and computation ability. Computationally intensive tasks will consume the battery much faster than ordinary tasks. Recently, a technology, so-called remote execution, becomes very popular [7]. By delegating some computation tasks to remote servers, the energy of mobile clients can be saved significantly. Some widely desired functions such as encryption/decyption, transcoding, require much more energy than ordinary packet sending and receiving, thus offloading these tasks to some trustworthy infrastructure will be very useful.

Such services can be deployed by either an ISP or a third-party service provider. If there is a PacketCloud site within the trust boundary of the mobile client, it can act as an infrastructure for on-path offloading. The sender will put the original payload in a PacketCloud-enabled packet, and deliver this packet to the chosen PacketCloud site. The computationally intensive tasks will be conducted there before arriving the destination. Please note that there may be more then one presence for the processing, which requires more than one PacketCloud sites with corresponding services deployed. The final presence will deliver the processed payload to the destination. With the help of PacketCloud, the communication time of a mobile device can be extended in scenarios such as creating a secure channel for VPN.

2.6.2 General Services

Protocol-Independent Cache There are lots of popular contents on the Internet, e.g., youtube video of a breaking news, tweets of a famous people, etc. These contents are repeatedly transferred across the Internet thus lead to wide existence of traffic redundancy. Similar to [24, 2], we are aiming at introducing a protocol-independent redundant traffic elimination scheme, working at the IP layer. Thus, redundant strings of bytes delivered by different higher layer protocol can be identified, cached, and delivered using an abstracted fingerprint for bandwidth saving.

An ISP can deploy a source-controlled caching service in different PacketCloud sites located at stub networks. We adopt share cache architecture proposed in [24] for redundancy elimination, which has been used as a building block in redundancy-aware routing [2] as well. Every packet travels through two PacketCloud sites, one ingress site nearby the sender and one egress site nearby the receiver. Every site pair utilizes a shared cache for frequently appeared contents by saving these information (fingerprint storage and packet store) in the FIS. The redundant contents are encoded at the ingress site and decoded at the egress site, thus the unnecessary traffic between these two sites can be saved. Since this service is source-controlled, the ISP may advertise lower price to encourage the more users to use the caching-aware packet.

Inter-domain Multicast: IP Multicast is an efficient data delivery manner, which can play a key role in different popular applications, such as massive multiplayer games, IPTV, conferencing. Although IP Multicast was proposed decades ago, there are still very few large scale deployments so far, especially in the inter-domain level. The only known feasible inter-domain IP multicast approach which supports multi-source applications is Free Riding Multicast (FRM) [22]. For every FRM-enabled domain, its BGP advertisements will

be augmented with a description of the FRM multicast groups it involves. Thus the group information can be propagated through BGP. To construct the multicast tree, the border router of the sender's domain scans its BGP table and computes the AS-level multicast tree based on the stored group information, which is the union of the BGP unicast paths to all destination domains.

By deploying a sender-controlled PacketCloud service, the required computation and storage tasks can be offloaded from the border routers, therefore can scale up to more multicast groups. The service can obtain the necessary group information from BGP table through an API. The sender can set the circuit label for identifying the multicast group, and forward the packet to the co-located PacketCloud site of the border router through unicast. After constructing the multicast tree, an additional shim header contains the constructed multicast tree will be inserted for further forwarding. With the help of local cache, duplicate FIS lookup and multicast tree computation can be avoided.

The local cache can help one server to perform the computation task for constructing the multicast tree once for each flow. After arriving the border router of every domain involving in the multicast group, the packet will be updated to an ordinary multicast packet and be delivered through intra-domain multicast.

Firewalls and Filters PacketCloud sites are natural locations to deploy in-network firewalls and filters to cut down malicious and unwanted traffic close to its sources. Traditionally, most unwanted traffic is filtered out at the destinations, because the ISPs usually do not know what traffic is wanted by different destinations. This can cause a waste of bandwidth, and increases the bandwidth costs of the popular Internet destinations. With PacketCloud and the destination-controlled services (§ 2.4), a destination can then take control of the traffic sent towards it at an early stage of the forwarding.

For instance, Google may want to deploy a distributed in-network firewall system to filter out the packets sent towards its data center that are non-TCP or have a destination port different from 80 or 443. It can create a simple PacketCloud service that checks the protocol field in the IP header and the destination port field in the TCP header. The service may also whitelist a number of source prefixes that belong to Google. The whitelist table can be stored in the FIS, so that Google can update it on-the-fly without re-deploying the service. Google then deploys the service as a destination-controlled service on the PacketCloud sites that are close to the malicious sources that send most of the unwanted traffic. Note that in this example, Google is both the PSP and the user of the service.

Besides simple firewalls, PacketCloud can also support the data plane design of the more complex filtering schemes such as StopIt [16]. Here the details of the filters can be stored in the FIS, and are dynamically adjusted by the control plane of such systems.

Evaluation of novel architectures PacketCloud is also a good environment for evaluating novel architectures. Either an ISP or a third-party entity can conduct such evaluations easily within the PacketCloud framework. Let us take Rule-Based Forwarding (RBF) [20] as an example, which is a new network design for flexible forwarding. RBF requires each packet to carry a *rule* in the packet header, which is a simple if-then-else construct to instruct a router how to forward the packet. RBF supports many different applications, such as source routing and middlebox traversal. To deploy RBF's data plane in PacketCloud, one only needs to deploy a sender-controlled service that works as an RBF rule interpreter. Every rule will be assigned a unique first hop circuit id. Also, the metadata field can be use to carry the state information.

3. ANALYSIS OF COST AND REVENUE

Deploying the PacketCloud platform will incur costs in various aspects, such as equipment, electricity, and IT staff. On the other hand, the platform can also generate revenue by leasing the resources to PSPs. In this section, we perform a simple cost analysis using the current market data to examine whether deploying PacketCloud is likely to be profitable for an ISP.

First, we estimate the yearly cost c of operating a PacketCloud site using the following formula:

$$c_{site} = N \times (c_{machine}/d) \times (E + 1) \quad (1)$$

N is the number of machines the ISP needs to provision in the computation cluster and in the storage service. N depends on the both expected amount of traffic that needs to be processed and the workload complexity. d is the depreciation time (years) of a machine, and $c_{machine}/d$ stands for the amortized yearly cost of a machine. E is an efficiency factor that shows how significant the remaining cost of running a site (renting, electricity, cooling, backup, IT staff, etc.) is compared to the machine cost.

To estimate N, we assume the site is fully saturated by a heavy service that plays the IPsec gateway role for every packet. We assume the maximum designed throughput of the site is 40Gbps. We use the commodity PacketShader server (CPU only) as an example, which consists of two quad-core Intel Xeon 5550 CPU, six DDR3 EEC 2GB RAM, one Super Micro X8DAH+F mainboard, and four Intel X520-DA2 dual-port 10GbE NIC. According to [14], it can achieve about 2.91 Gbps throughput for 64B IPsec packet. In such case, $40Gbps/2.91Gbps \approx 14$ machines are needed. The current market price of such a machine is about \$4128[1]. with a normal depreciation time of 3 years ($d = 3$). Recent cost analysis [12] of modern data center suggests that the machine cost is almost the same as the aggregation of all other costs, and therefore $E \approx 1$. In the final calculation of c, we assume 10% more investment should be added for the

[1]Prices are from Google Checkout on July 2011

storage service and the load balancer. This gives us the final estimated yearly cost c to operate a PacketCloud site.

$$c_{site} = (14 \times 1.1) \times (4128/3) \times (1+1) \approx \$42.38K/yr \quad (2)$$

Since we are aiming at hosting different services on the same physical node, the multiplexing will introduced some extra overhead for multi-service hosting, which makes the implementation critical to the final cost calculation. Anyway, Eq.(2) still gives us a ballpark investment estimation of a site.

There are two revenue sources for an ISP by operating a PacketCloud site. One source is by hosting several in-network services itself, which can change the economic incomings. For example, deploying mobile services which can enhance the user experiences of mobile device users, thus larger user amount and more activities are desirable. Also, the deployment of in-network cache can reduce the redundant traffic in the routing infrastructure, thus the users can get better data delivery experience in return. Both of these are helpful for making the ISP services more attractive, and economically sound. The other source is by leasing some resources to third-party service providers. Let us do a very simple profit calculation for an ordinary computation node.

In the estimation of the revenue, we only consider the CPU cost as it is likely to be the most heavily consumed resources. The CPU cost means the CPU cycles consumed by a service for on-path packet processing (e.g., encryption), API calling (e.g., FIS reading/writing, information logging), and packet forwarding. We are not considering the revenue of network bandwidth in this simple calculation.

Since a PacketCloud site is similar to a PaaS cloud platform, we borrow the CPU price published by Google AppEngine [11], a popular PaaS for web applications. AppEngine charges $\$0.10$ per CPU hour for a single core 1.2GHz Intel processor. We first scale it to the $2.66GHz$ CPU used above, and therefore the revenue is $0.10 * (2.66/1.2) = \$0.22$ per hour per CPU core. For one computation node which has 8 cores (2 quad-core CPUs), the maximum yearly CPU revenue for a computation node is $r_{CPU} = 0.22 \times 8 \times 24 \times 365 = \$15.42K$. Comparing with a computation node's yearly cost $4128/3 \times (1+1) = 2.75K$, even 10% of the maximum revenue is still comparable.

4. RELATED WORK

Proposed more than a decade ago, Active Network [27, 30] allows routers and switches to perform customized computational tasks on the packets traversing them. The packets can be modified and/or redirected to a different next hop. PacketCloud shares the similar vision as Active Network. We focus on building a practical and economic-viable platform that can host various types of in-network services.

Some in-network services such as Internet Indirection Infrastructure (i3) [26], end-to-end abstractions [21], and VINI [5] have taken the overlay network approach to avoid the deployment challenge in the real Internet. However, overlay networks are known to have performance and reliability problems, because the overlay nodes are usually end systems with heterogeneous access speed. Also, the overlay network may generate some unwanted traffic patterns for the ISPs. With PacketCloud, the services can be deployed inside the real Internet on reliable ISP-managed PacketCloud sites with high speed links between them. It is our future work to implement and evaluate the performance of these services when deployed in PacketCloud.

Programmable routers can be a potential way to host in-network services. With the development of commodity PCs, some newly proposed software routers such as RouteBrick [9] and PacketShader [14] can achieve high forwarding throughput. On the other hand, researchers have also proposed new hardware router designs such as SwitchBlade [3] and NetFPGA [18] using programmable hardware platforms such as FPGA. Deploying the new programmable routers requires the ISPs to phase out the legacy ones, and the process can be costly. Further, it is unclear why the ISPs have the incentives for the upgrade. In contrast, PacketCloud decouples between a router and its co-located PacketCloud site. The scheme is backward compatible with legacy routers. Also, because of the decoupling, we can focus on benchmarking and optimizing the services on separate hardwares, while letting the existing routers play their ordinary duties. Further, the physical separation between the two entities can minimize the interference between the routing tasks and the packet processing tasks, and can also strengthen security.

SideCar [23] aims to build programmable datacenter networks with legacy switches and commodity servers. Sharing the same decoupling idea as PacketCloud, Sidecar allows switches to send sampled packets to the commodity SideCar servers for further processing. In contrast, PacketCloud focuses on bringing programmability to the Internet core instead of datacenter networks. Further, in PacketCloud all packets are forwarded to a PacketCloud site for processing, instead of just a sampled version.

Public cloud platforms such as Amazon's AWS [1] and Google's AppEngine [11] are places to deploy end-to-end network services such as web applications. A PacketCloud site shares many features with these platforms, such as the ability to host multiple services and to scale up automatically. We also borrow the pay-as-you-go charging model, especially the one of AppEngine which charges by real CPU usage instead of the virtual machine leasing time. On the other hand, the internals of a PacketCloud site are very different from those of a public cloud datacenter. First, a PacketCloud site may only contain hundreds of CPU cores (§ 3), while a public cloud datacenter can be much larger. Second, the components in a PacketCloud are customized for the packet processing workload, such as the APIs and the FIS, while a public cloud is built to support more generic workload. Third, since the number of public data centers are still relatively few in number, this centralization trend also in-

creases the data center distance to the users [29]. Therefore, hosting the in-network services on public cloud platforms will introduce data delivery detour significantly, which is harmful to latency-sensitive applications.

5. CONCLUSION AND FUTURE WORK

In-network services such as DoS protection and redundant traffic elimination are desirable as they can better manage the traffic inside the core of Internet. Also, offloading computational intensive tasks to in-network services is very attractive for mobile devices users. However, deploying the services in the current Internet is challenging, because the existing routers have little programmability, and replacing them with the new programmable ones is costly and does not have enough incentives for the ISPs. This paper proposes PacketCloud, a new way to host in-network services inside commodity packet processing clusters that are co-located with the legacy routers. PacketCloud does not require router upgrades, and has a built-in economic-model that allows ISPs to gain revenue by selling their computation and bandwidth resources to the third-party service providers. Our preliminary cost analysis suggests that deploying PacketCloud can be profitable for an ISP with a moderate utilization ratio. We further evaluate the generality of the platform using a few case studies.

As future work, we plan to implement a prototype of a PacketCloud site using commodity equipment, and evaluate its performance using real in-network services. Meanwhile, many other interesting issues can be explored further such as what is the detailed billing policy for ISPs and PSPs, how to use network layer information for new services, how to incorporate new hardwares (e.g., GPU, NetFPGA), how to provide a socket interface for the end clients (i.e., PacketCloud-enabled TCP/UDP) and how to model the potential abuses and attacks of the platform.

Acknowledgement

This material is based upon work supported by the National Science Foundation under Grant No. 1040043.

6. REFERENCES

[1] Amazon Web Service. http://aws.amazon.com/
[2] A. Anand, A. Gupta, A. Akella, et al. Packet Caches on Routers: The Implications of Universal Redundant Traffic Elimination. Proc. of ACM SIGCOMM, 2008.
[3] M.B. Anwer, M. Motiwala, et al. SwitchBlade: A Platform for Rapid Deployment of Network Protocols on Programmable Hardware. ACM SIGCOMM, 2010.
[4] M. Armbrust, A. Fox, et al. A view of Cloud Computing. Communications of the ACM, 2010, 53(4): 50-58.
[5] A. Bavier, N. Feamster, M. Huang, et al. In VINI Veritas: Realistic and controlled network experimentation. Proc. ACM SIGCOMM, 2006.
[6] Cisco white paper. Policy-Based Routing.
[7] E. Cuervo, A. Balasubramanian, et al. MAUI: Making Smartphones Last Longer with Code Offload. Proc. of ACM Mobisys, 2010.
[8] V. Devarapalli, R. Wakikawa, et al. Network Mobility (NEMO) Basic Support Protocol. RFC 3963.
[9] R. Dingledine, N. Mathewson, et al. Tor: The Second-Generation Onion Router. Proc. of USENIX Security, 2004.
[10] M. Dobrescu, N. Egi, et al. RouteBricks: Exploiting Parallelism To Scale Software Routers. Proc. of SOSP, 2009.
[11] C. Estan, A. Akella, and S. Banerjee. Achieving Good End-to-End Service Using Bill-Pay. Proc. of ACM HotNets, 2006.
[12] Google App Engine. http://code.google.com/appengine/ .
[13] A. Greenberg, J. Hamilton, et al. The cost of a cloud: research problems in data center networks. ACM SIGCOMM CCR, 2009, 39(1):68-73.
[14] A. Greenhalgh, F. Huici, et al. Flow Processing and the Rise of Commodity Network Hardware. ACM SIGCOMM CCR, 2009, 39(2):20-26.
[15] S. Han, K. Jang, et al. PacketShader: a GPU-accelerated Software Router. Proc. of SIGCOMM, 2010.
[16] A. Li, X. Liu, X. Yang. Bootstrapping Accountability in the Internet We Have. Proc. of NSDI, 2011.
[17] X. Liu, X. Yang, and Y. Lu. To Filter or to Authorize: Network-Layer DoS Defense Against Multimillion-node Botnets. Proc. of ACM SIGCOMM, 2008.
[18] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek. The click modular router. SIGOPS Oper. Syst. Rev., 33(5):217-231, 1999.
[19] J. Naous, G. Gibb, et al. NetFPGA: reusable router architecture for experimental research. Proc. of ACM PRESTO, 2008.
[20] C. Perkins. IP Mobility Support. RFC 2002.
[21] L. Popa, N. Egi, et al. Building Extensible Networks with Rule-Based Forwarding. Proc. of OSDI, 2010.
[22] S. Shanbhag, T. Wolf. Implementation of End-to-End Abstractions in a Network Service Architecture. Proc. of ACM CoNEXT, 2008.
[23] S. Ratnasamy, A. Ermolinskiy, S. Shenker. Revisiting IP Multicast. Proc. of ACM SIGCOMM, 2006.
[24] A. Shieh, S. Kandula, E.G. Sirer. Sidecar: Building programmable datacenter networks without programmable switches. Proc. of HotNets, 2010.
[25] N.T. Spring and David Wetherall. A protocol-independent technique for eliminating redundant network traffic. Proc. of ACM SIGCOMM, 2000.
[26] S. Srinivasan, J. W. Lee, E. Liu, et al. NetServ: Dynamically Deploying In-network Services. Proc. of ACM ReArch, 2009.
[27] I. Stoica, D. Adkins, S. Zhuang, et al. Internet indirection infrastructure. IEEE/ACM Trans. Netw., 2004, 12(2):205-218.
[28] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, et al. A Survey of Active Network Research. IEEE Communications Magazine, 1997, 35(1):80-86.
[29] V. Valancius, C. Lumezanu, N. Feamster, et al. How Many Tiers? Pricing in the Internet Transit Market. Proc. of ACM SIGCOMM, 2011.
[30] V. Valancius, N. Laoutaris, et al. Greening the Internet with Nano Data Centers. In Proc. of ACM CoNEXT, 2009.
[31] D. Wetherall. Active Network Vision and Reality: Lessons from a Capsule-Based System. Proc. of SOSP, 1999.