PacketCloud: A Cloudlet-Based Open Platform for In-Network Services

Yang Chen, *Senior Member, IEEE*, Yu Chen, Qiang Cao, and Xiaowei Yang

Abstract—The Internet was designed with the end-to-end principle where the network layer provided merely the best-effort forwarding service. This design makes it challenging to add new services into the Internet infrastructure. However, as the Internet connectivity becomes a commodity, users and applications increasingly demand new in-network services. This paper proposes PacketCloud, a cloudlet-based open platform to host in-network services. Different from standalone, specialized middleboxes, cloudlets can efficiently share a set of commodity servers among different services, and serve the network traffic in an elastic way. PacketCloud can help both Internet Service Providers (ISPs) and emerging application/content providers deploy their services at strategic network locations. We have implemented a proof-of-concept prototype of PacketCloud. PacketCloud introduces a small additional delay, and can scale well to handle high-throughput data traffic. We have evaluated PacketCloud in both a fully functional emulated environment, and the real Internet.

Index Terms—Cloud computing, in-network services, open platform, elasticity

1 Introduction

The Internet was designed with the end-to-end principle where the network layer offered merely the best-effort forwarding function. Many important communication functions such as reliability and state management are placed at the end points [1]. This design provides low complexity and high robustness at network routers, and is necessary to meet the technology constraints of the 1970s when the Internet was launched. However, nowadays the Internet connectivity has become a commodity, users and applications increasingly demand new in-network services. Deploying in-network services can either enhance the user experience while surfing the Internet, or optimize the network traffic. Prospective in-network services include host mobility support [2], efficient delivery for frequently accessed content [3], on-path storage [4], or malicious traffic filtering [5].

To host in-network services, middleboxes [6] are widely used. In today's Internet, ISPs have deployed a number of middleboxes such as content caches, firewalls, and load balancers. Besides the ISPs, emerging third-party providers such as Internet-based content providers and application providers are also interested in getting their services close to end users. As they do not own the Internet infrastructure, they need to collaborate with the ISPs to deploy their middleboxes. Many of existing middleboxes are standalone, specialized hardware [7], [8]. Therefore, there are three significant problems. First, available resources of co-located middleboxes cannot be shared among different services. Second, the capacity of each type of middleboxes needs to be sufficient to handle the peak traffic. This leads to over-provisioning during off-peak hours. Third, hardware-based middleboxes are hard to upgrade [7], as new functions cannot be added easily.

We aim to make the Internet infrastructure as an open platform to host in-network services, and avoid the inflexibility of using middleboxes. We propose an architectural solution, called PacketCloud. PacketCloud allows ISPs to integrate "cloudlets" into the network. A cloudlet is a general-purpose cluster consisting of a set of commodity servers, and it can host in-network services. ISPs can choose suitable Point of Presences (PoPs) to deploy cloudlets. In each cloudlet, commodity servers can be efficiently shared among different services, and accordingly an elastic resource allocation can be achieved. PacketCloud supports Linux-based general-purpose services, and it is open to third-party service providers, including content providers and application providers. PacketCloud can host a number of viable in-network functions, such as fast content fetching, privacy preserving, and energy saving for mobile devices.

As an open platform, PacketCloud not only helps end users, but also offers viable benefits to both ISPs and third-party application/content providers. PacketCloud enables the ISPs to tailor the network traffic. By placing traffic optimizers/firewalls at data forwarding paths, duplicated/malicious traffic can be efficiently reduced. For third-party service providers, PacketCloud allows them to deploy their services in selected in-network cloudlets, by renting resources from corresponding ISPs. This helps end users access these services within their local ISPs. In short, the use of PacketCloud will not only provide a cost-effective replacement of existing middleboxes, but also get third-party application/content providers close to end users. For example, Netflix, which consumes more than 30 percent of peak

- *Y. Chen (Yang Chen) is with the School of Computer Science, Fudan University, China, and the Engineering Research Center of Cyber Security Auditing and Monitoring, Ministry of Education, China. E-mail: chenyang@fudan.edu.cn.*
- *Y. Chen (Yu Chen), Q. Cao, and X. Yang are with the Department of Computer Science, Duke University. E-mail: {yuchen, qiangcao, xwy}@cs.duke.edu.*

Manuscript received 20 Sept. 2014; revised 27 Feb. 2015; accepted 11 Apr. 2015. Date of publication 16 Apr. 2015; date of current version 16 Mar. 2016. Recommended for acceptance by Y. Cui.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2015.2424222

traffic on US ISP networks, begins to deliver customized content caching middleboxes to ISPs for traffic optimization [9]. If PacketCloud was deployed by related ISPs, Netflix could directly rent the resources from selected cloudlets to host their content caches, instead of involving in the tedious delivery and remote management of physical servers. Accordingly, ISPs can earn viable economic rewards by offering the cloudlet resources.

We have built a proof-of-concept prototype of PacketCloud, and several representative use cases. We have four key findings in our evaluation.

- We evaluate PacketCloud in a fully functional, emulated environment. The results show that PacketCloud is scalable and can handle high-throughput data traffic, while only introducing a small per-hop delay penalty.
- We conduct an Internet-based latency evaluation by comparing different possible locations for hosting in-network services, i.e., on default end-to-end paths, Amazon EC2 data centers, and in-network cloudlets. Our results show that using in-network cloudlets can achieve the shortest "sender-service-receiver" delay.
- Using the PlanetLab testbed, we build an Internet-based content delivery application to show that PacketCloud is beneficial for different Internet entities, including end users, ISPs, and third-party service providers.
- We implement and evaluate an Intrusion Detection System (IDS) service using both an emulated environment and the real Internet. We can see that PacketCloud is very helpful to filter malicious packets out.

The rest of the paper is organized as follows. In Section 2, we introduce the design rationale. In Section 3, we present the PacketCloud system design. In Section 4, we demonstrate how to address the key challenges in implementing PacketCloud. In Section 5, we evaluate PacketCloud from various aspects. Several viable issues are discussed in Section 6, and related work are listed in Section 7. Finally, we conclude the whole paper in Section 8.

2 Design Rationale

As proposed in [10], one approach for hosting in-network services is to use public data centers. In this section, we want to demonstrate why we propose to use cloudlets to host in-network services, instead of using public data centers.

Building a public data center is expensive, given the large investment on computing servers, routing devices, and cooling systems. In addition, to operate a data center, the cost of electricity power is significant. Therefore, to find a location to establish a new data center, there are a number of issues to consider, such as the user distribution, the power energy reliability, and the price of local electricity power. As a result, even the world's leading public cloud providers own very few data centers. For example, the Amazon EC2 cloud has only eight data centers all over the world. Differently, in PacketCloud, the small size of the cloudlets allows ISPs to deploy in-network services in a number of network locations, e.g., PoPs. Even a small ISP can deploy multiple cloudlets in its PoPs. These in-network services can be closer to end users than public data centers.

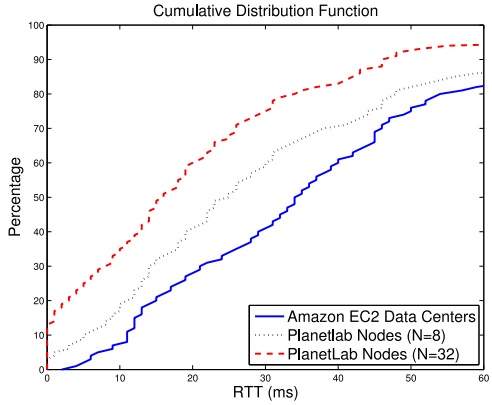


Fig. 1. Distribution of distances to Cloudlets/EC2 data centers.

In this section, we plan to use an Internet-based measurement study to see the difference between cloudlets and public data centers, in terms of data delivery latency. We pick 580 PlanetLab nodes all over the world to act as end users, and most of them belong to universities. We consider the following three sets of service hosting nodes.

- Set A: eight virtual instances (each from one of the eight Amazon EC2 data centers)
- Set B: eight randomly selected PlanetLab nodes
- Set C: 32 randomly selected PlanetLab nodes

Set A covers all network locations of the Amazon EC2 data centers. Set B allocates equal number of nodes as set A, while the selected nodes belong to universities. Therefore, we can see the difference between deploying services on EC2 data centers, and on cloudlets located in universities. Set C has more nodes than set A/B, and accordingly we can see the performance gain of deploying services in more locations. Note that such increase of locations is only feasible by using cloudlets, as building an additional data center for a public cloud provider is extremely difficult due to the high cost. For every end user u, we define the network distance between itself and a set S as the round-trip delay between the end user and the closest node in S, i.e., $d = min_x(RTT(u,x)), x \in S$.

Fig. 1 shows the cumulative distribution function (CDF) of the network distances between all end users and the sets A, B, and C. The median value of the network distances between the end users and the sets A, B, and C are 34, 23, and 13 ms, respectively. Therefore, using cloudlets can introduce more flexibility in hosting services in more network locations, and get services closer to end users than using public data centers.

Beside the short access latency and flexible deployment, the use of cloudlets can avoid some unwanted inter-domain data traffic, and data ownership problems [11]. Also, in terms of cost-effectiveness, building cloudlets is better than renting resources from public data centers. We use Amazon EC2 service as an example. One typical instance type on EC2 is known as *m3.2xlarge* instance, which has eight virtual CPUs, 30 GB Memory, and two 80 GB SSD disks. The CPU model of the corresponding physical server is Intel(R) Xeon (R) CPU E5-2670, running at 2.50 GHz. According to the

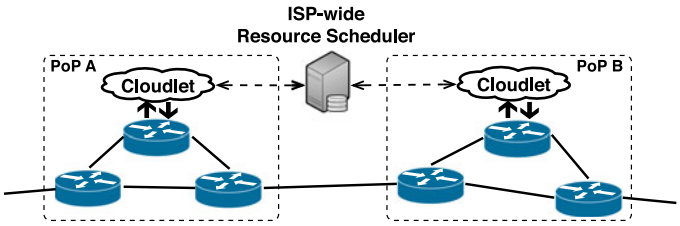


Fig. 2. Cloudlets in an ISP.

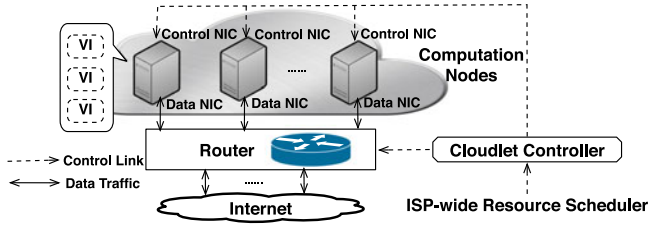


Fig. 3. Building blocks of a Cloudlet.

price we retrieved on February 4, 2015, renting such an instance from the North Virginia data center requires $0.560 per hour, i.e., $4905.6 per year. For comparison, we need to calculate the price of buying a physical server with the same (or slightly better) hardware standard. By referring to the *Amazon.com* platform, we can see that, an Intel(R) Xeon (R) CPU E5-2670 costs $1420, an Intel(R) Server Board S2600CP2 costs $503, a pair of 16 GB RAM stick costs $266, a pair of 120 GB SSD disks costs $106, and a Logisys computer case with power supply costs $30. Note that the server board has an Embedded Intel I350 Dual Gigabit Adaptor. We use these five components to build a server, and the total price is $2325. We assume the depreciation time is three years, and accordingly the yearly cost is $775. We are aware that some moderate additional costs such as electricity power, cooling, and IT staff are needed. Nevertheless, we believe that building cloudlets will still not be as expensive as renting resources from public data centers.

3 System Design

This section describes the design of PacketCloud. We provide a list of design goals in Section 3.1. We give an overview of PacketCloud architecture in Section 3.2. PacketCloud supports two types of services, and we describe both of them in detail in Section 3.3. As the deployed cloudlets form an elastic resource pool, we investigate the resource allocation and adjustment policies in Section 3.4 and Section 3.5. We discuss reliability and security issues in Section 3.6.

3.1 Design Goal

Before stepping into the detailed design of PacketCloud, a list of design goals is shown here:

- Elasticity. To serve the highly dynamic network traffic, PacketCloud is designed to host elastic in-network services. In other words, we allow a deployed service to scale up and down according to the traffic demand. During peak hours, the service should be able to allocate enough resources to handle the data traffic. During off-peak hours, the service can release unnecessary resources.
- Efficiency. Different from specialized middleboxes, available resources are required to be shared among different services.
- Reliability and Security. As an open platform, security is very important as deployed services might be unstable, or even malicious. PacketCloud must be robust to different kinds of service failures and malicious attacks.

3.2 Overview

To deploy PacketCloud, we assume that an ISP has built p in-network cloudlets, called $C_1, C_2, \ldots, C_p$. These general-purpose cloudlets can be placed at selected PoPs. The ISP aggregates all the p cloudlets as a resource pool. We use a logically centralized *ISP-wide resource scheduler* to manage all these cloudlets. As shown in Fig. 2, this resource scheduler manages all deployed cloudlets. Every cloudlet sends a resource report to the resource scheduler periodically, including the reserved and available resources (CPU, memory, disk, and network). Therefore, the resource scheduler maintains a local database to keep an aggregate view of the resources of all p cloudlets. The resource scheduler also serves as the sole interface for cloudlet resource allocation. In-network service providers, including the ISP itself, and third-party service providers, should contact the resource scheduler for resource reservation.

A cloudlet is a functional unit in the resource pool. As shown in Fig. 3, to exchange data with the Internet, every cloudlet has a co-located *front-end router* (or switch). Besides acting as an integrated part of the global routing system, the router checks against every packet, and see if it is related to any deployed in-network service on the co-located cloudlet. If so, the packet will be forwarded to the cloudlet for further processing.

A cloudlet is composed of a *cloudlet controller*, and a number of commodity servers, called *computation nodes*. The cloudlet controller is a general-purpose server, and is responsible for the resource allocation and management of all nodes within the cloudlet. The cloudlet controller actively monitors the resources of every computation node, and reports to the ISP-wide resource scheduler periodically.

The computation nodes are used for hosting in-network services. To efficiently utilize the resources, we allow one computation node to host multiple services at the same time. Since PacketCloud is an open platform to host services from different service providers, the deployed services could be experimental, and might crash at any time. Virtualization is necessary to provide isolation among them, and a computation node can host multiple virtual instances (VIs) simultaneously.

PacketCloud supports multiple types of VIs. Each type has a specified resource capacity. A VI can host general-purpose in-network services subject to its resource constraint. A service provider can reserve any VI that meets its requirement. The ISP can also suggest a cost-effective VI type for the service provider based on benchmarking results [12], e.g., how much traffic an intrusion detection system can inspect per second using a certain type of VI. To handle high-throughput traffic, a service provider can even reserve multiple computation nodes.

The resources of every computation node can be represented by a four-tuple $< c, m, d, n >$, including the resource capability of CPU, memory, disk, and network. The cloudlet controller keeps track of the total resources T and available resources A of every computation node. At time t, the total resources of a computation node can be denoted as $T(t) =< c_T(t), d_T(t), m_T(t), n_T(t) >$, and the available resources of this node can be denoted as $A(t) =< c_A(t), d_A(t), m_A(t), n_A(t) >$.

3.3 Service Types

To satisfy different Internet entities including end users, ISPs, and third-party providers, PacketCloud allows deployed in-network services to be activated in two different ways. On one hand, a service provider can let the public know the network address of a deployed service. This type of services are denoted as *visible services*. On the other hand, a service provider can choose to sniff or even modify the traffic on-the-fly, without the awareness of end users. This type of services are invisible to end users, and accordingly, we call them *transparent services*. Let us go into the details of these two types of services.

3.3.1 Visible Services

A visible service is publicly known. For example, an ISP can deploy a local weather information service, and an end user can use it to obtain the latest weather forecast information. Such service will be hosted by a dedicated VI, and a unique network address will serve as the identifier of this service. Demultiplexing traffic related to a visible service is simple, since such traffic is distinguishable by the destination IP address of every data packet. The routers only need to support legacy routing and forwarding functions.

For every deployed visible service, the service provider is responsible to announce the corresponding network address to the public. They might also choose to register some meaningful domain name to identify a deployed service. For example, a domain name like *cloudlet-location.service-name.org*. Particularly, to serve end users from different networks, a service provider might reserve PacketCloud VIs from different ISPs, and let these VIs offer the same service function. The service provider can register a single domain name, and collaborate with local ISPs to redirect the requests to this domain name to a local service agent. An example is the local weather service. The service provider can just register a domain name like *local-weather.org*, and every user can reach its closest agent for weather forecast by referring to the same domain name.

Some visible services are shown as follows:

- *Mobile offloading*. Nowadays mobile devices are widely used, and these devices always have a limited battery life and computation capability. By using PacketCloud, computationally intensive tasks can be offloaded to in-network cloudlets, and the battery energy can be saved.
- *Delay-tolerant content delivery*. To handle network congestion, upgrading the capacity of bottleneck links can be expensive, not to mention the unavoidable over-provisioning during off-peak hours. Alternatively, ISPs may want to host some in-network storage instances [13] to store delay-tolerant contents. An end user can choose to send data through a topologically-closest storage instance. The data will be stored temporarily if the path between the instance and the destination is congested. When the network becomes less loaded, the content can be sent out to the destination. This approach leaves room for time-sensitive traffic.
- *Anonymous communication*. Anonymous communication is a vital way for anti-censorship, and onion routing is a popular method. Today's end hosts-based Tor overlay suffers from heterogeneous access latency of end hosts [14]. With PacketCloud, onion routers can be hosted by in-network cloudlets to attain good network connectivity. As the network addresses of these onion routers are publicly known, a data sender can pick some of them as the intermediary nodes, and send encrypted messages to the first node. Each on-path onion router can remove a layer of encryption to learn which is the next node on the forwarding path. To avoid ISP-based traffic observation, we can adopt the algorithm in [15] to improve the location diversity while building a forwarding path.
- *Location-based services*. The rapid development of mobile devices also motivates the wide usage of location-based services. Such services can help mobile users to receive local weather reports, find nearby bus stops, explore new places, and communicate with other local users. The service provider might reserve a number of VIs at different cloudlets, and deploy service agents on these VIs. For the convenience of end users, the service provider can utilize a unique domain name to represent all service agents, e.g., *service-name.org*. For any end user, the local ISP will translate this domain name into the network address of its closest VI.

3.3.2 Transparent Services

Different from visible services, transparent services cannot be activated by end users. Instead, they are triggered by the ISPs by intercepting and further processing legacy end-to-end packets on-the-fly. The traffic sender and the desired receiver may not even be aware of the existence of these services. The ISPs, or third-party service providers, can place transparent services on the cloudlets located at strategic network locations, i.e., path aggregate points, to integrate services to the default packet forwarding paths. PacketCloud allows a deployed transparent service to add some pre-configured *demultiplexing rules* (e.g, a specified source/destination network address, a specified port number in the packet header) to the front-end router, to identify packets related to this service. To ensure the compatibility among different hardware vendors, we pick the widely used OpenFlow protocol as the basic rule specification language. Accordingly, the configured router can intercept selected packets and forward them to a co-located transparent service without notifying the sender and the receiver. For example, a demultiplexing rule of "destination tcp port = 80" lets the router

TABLE 1
A Sample Cloudlet Resource Request

Time slot	$[t_0, t_1]$
Requested VI capacity	$\langle c_r, d_r, m_r, n_r \rangle$
Desired cloudlet (optional)	C_x

TABLE 2
A Sample Request with a Varying Capacity

subslot	desired capacity
$[t_0, t_M]$	$< c = 1, m = 256\,MB, d = 300\,GB, n = 1\,Gbps >$
$[t_M, t_1]$	$< c = 2, m = 1\,GB, d = 900\,GB, n = 5\,Gbps >$

intercept all the HTTP traffic it receives, and redirect the intercepted packets to a transparent service.

We enumerate the following cases of transparent services.

- *IPSec tunnel*. An organization may have multiple branches in different locations. Confidential and unencrypted data may go through untrusted networks between two branches. Using PacketCloud, the two branches can transparently deploy an encryption service for outgoing traffic, and one decryption service for incoming traffic. This IPSec tunnel secures the inter-branch data transmission in an automatic way.
- *WAN Optimizer*. People might transfer uncompressed data through the Internet. Such data traffic can be represented more efficiently using compression technologies. An ISP can perform WAN optimization between two strategic network locations at the data forwarding path. The redundant content can be compressed before the delivery between these two locations. Such compression would become quite helpful while data traffic has to travel through some network bottleneck.
- *Intrusion detection system*. An ISP's PoPs are natural locations to deploy in-network IDSes to identify malicious/unwanted traffic. ISPs can dynamically adjust the filtering policies, for example, some bit patterns in the packet header. Identified traffic will be either blocked, or rate limited.
- *Load balancer*. A load balancer is a proxy to distribute the workload across multiple servers with a shared publicly known IP address. The load balancer can divide all packets going to this address into multiple flows, each of which will be redirected to a different server.
- *Network monitoring*. ISPs can perform network monitoring through PacketCloud. Deployed monitoring agents can sniff the up-to-date network information, and perform some pre-processing. Also, the ISPs can further analyze the aggregate data collected from various network locations.
- *Transparent web caching*. To enhance content delivery efficiency, caches can be deployed at path aggregate points, keeping local copies of frequently requested contents. Caches might be deployed by ISPs or third-party content providers.

3.4 Resource Allocation

Every ISP uses an ISP-wide resource scheduler to manage its PacketCloud services. Both the ISP itself and all third-party providers need to reserve a VI to host the desired service through this scheduler. Also, it keeps an aggregate view of the resource availability information of all cloudlets, by receiving reports from the cloudlet controllers periodically.

To reserve cloudlet resources from an ISP, a reservation request should be filed to the scheduler. This request should include a requested time slot, a desired VI capacity, and optionally, a preferable cloudlet C_i. A sample request is shown in Table 1. PacketCloud also allows a request of a varying capacity within the desired time slot. On one hand, the investment of the service providers can be saved, as they do not need to request for unnecessary capacity during off-peak hours, and accordingly they can choose to just keep minimal resources. On the other hand, the cloudlet resources can be efficiently used. A service provider can split the whole time slot into a number of continuous sub-slots. In every sub-slot, a desired resource capacity can be specified. Table 2 shows an example for a request with a varying capacity for two continuous time slots.

Upon receipt of a request, the ISP-wide resource scheduler checks the local database to list the computation nodes that satisfy the request. For a computation node N, if for $\forall$ t $\in [t_0, t_1]$, we have $c_r(t) \leq c_A(t)$,$m_r(t) \leq m_A(t)$,$d_r(t) \leq d_A(t)$, and $n_r(t) \leq n_A(t)$, the node N could be a prospective computation node to host the service.

On one hand, if every cloudlet has no computation node with enough available resources, the request will be declined. On the other hand, there might be multiple computation nodes satisfying the request. In such a case, the ISP-wide resource scheduler will pick one node from the cloudlet which is topologically closest to the requestor. The scheduler further checks with the corresponding cloudlet controller to ensure the selected cloudlet is able to host the service. Note that the confirmation procedure can avoid possible information inconsistency, e.g., discovering some unexpected server crash occurs after the latest resource report was sent.

After a VI is reserved, the service provider can upload the service program to the VI. PacketCloud allows the service providers to upload general-purpose Linux-based programs.

The cloudlet controller releases the resources occupied by a certain service after the reservation slot expires. According to the record provided by the ISP-wide resource scheduler, the third-party content/application providers need to pay the corresponding ISPs in a "pay-as-you-go" fashion.

To serve customers around the world, a service provider might need to reserve cloudlet resources from multiple ISPs. For example, a content provider might want to deploy content caches in different ISPs. Similarly, a Tor service provider should ensure that Tor nodes are operated by different ISPs, otherwise the only ISP can easily find out the sender/receiver pair. To launch a cross-ISP service, the service provider can communicate with each relevant ISP for resource reservation, and the reserved instances will form a cross-ISP resource pool.

3.5 Resource Adjustment

PacketCloud allows a service provider to scale up and down the capability of a reserved VI. If a service provider wants to allocate more resources for a reserved VI, it needs to file an application to the ISP-wide resource scheduler, and the scheduler will check the resource availability with the corresponding cloudlet controller. If there are enough available resources, the request will be forwarded to the corresponding computation node. As the dynamic allocation of resources such as CPU and memory is a built-in feature of LinuX Containers (LXC), the capacity of the VI can be increased conveniently even when the service is running

Alternatively, a service provider might request to allocate fewer resources for a reserved VI. Although the cost of these released resources could be refunded, PacketCloud requires the service provider to pay a fine for such rescheduling. We encourage the service providers to cancel reservations earlier by giving them more refunds. We assume that a service provider reserved a VI at time t_A, and has paid a fee of F. This service is expected to start at time t_B. If the service provider would like to cancel the reservation at t_C ($t_A < t_C < t_B$), the money it can get back can be calculated as $F \times \frac{t_B - t_C}{t_B - t_A}$. According to this equation, the earlier the service provider decides to cancel the reservation, the more refund it will receive. If a service provider want to cancel a reserved service at the last moment, it will not receive any refund.

For a certain computation node, *scaling conflict* occurs when one or multiple VIs are demanding new resources, while the node is not able to allocate enough resources for them. We suggest to use some migration-based conflict handling method [16], i.e., by migrating some VIs to the least loaded node of the cloudlet. We leave the detailed conflict handling policy as a future work.

3.6 Reliability and Security

Both reliability and security are very important for PacketCloud as it works inside the Internet infrastructure. There are several issues we need to consider in the design.

Unexpected service failures. To limit the impact of unexpected failures, PacketCloud uses LXC-based virtualization for service isolation. Therefore, any unexpected crash will only fail the corresponding service within the VI instead of affecting other services running on the same computation node. Moreover, the physical decoupling between the cloudlet and the router ensures that the router will operate normally even when some computation node fails. This feature is important for deploying new experimental services.

Malicious demultiplexing rules. To decide which packets should be forwarded to a deployed transparent service, the service provider needs to provide some demultiplexing rules to the corresponding ISPs. A malicious rule can demultiplex some legitimate traffic, sniff the traffic, and even discard some data packets intentionally. To prevent from such attacks, PacketCloud inspects every third-party provider's demultiplexing rule. If a demultiplexing rule includes some destination prefix, the corresponding provider needs to sign up with the ISP with a 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 [17].

Malicious services. A third-party service may include some malicious functions in its code. To secure the whole cloudlet, PacketCloud only allows the corresponding ISP to own the root permission of every computation node. As every deployed service is restricted to be executed inside a VI, the ISP is able to inspect any packet going to/coming from the VI. Similar to [18], the ISP might choose to only inspect a sampled subset of these packets when the traffic is high. If there is any abnormal activity, the ISP can restrict the resources of a suspected service, or even terminate the service.

Excessive resource usage. Although PacketCloud is designed for handling high-throughput traffic, still, a cloudlet's processing capability is not as high as a public data center. Deployed computation resources need to be utilized efficiently. To reduce the total amount of available resources, a malicious third-party provider might want to occupy a significant part of a cloudlet with a low financial cost. To prevent from this, our solution is two-fold. On one hand, a fixed amount of dedicated resources are reserved for services deployed by the ISP who owns the cloudlet. On the other hand, we introduce a tiered pricing policy [19]. A high price will be applied to punish the third-party providers who intend to occupy an unreasonable amount of resources.

4 Implementation

In this section, we demonstrate how we implement PacketCloud. We go through the building blocks of PacketCloud in Section 4.1. We describe how a packet will be demultiplexed from the front-end router and processed by the cloudlet in Section 4.2. We list the services we have built and tested in Section 4.3.

4.1 Building Blocks of a Cloudlet

In this section, we focus on how to implement the building blocks of a cloudlet. Different from conventional data centers, which might have multiple levels of routers/switches, a cloudlet is much smaller. Typically, a cloudlet has one cloudlet controller, tens of computation nodes, and a co-located front-end router. We discuss the cloudlet controller in Section 4.1.1, and the computation nodes in Section 4.1.2. Particularly, we investigate the requirements for a cloudlet's co-located front-end router in Section 4.1.3.

4.1.1 Cloudlet Controller

The cloudlet controller has three functions. First, it is responsible for configuring the flow tables of the front-end router using the OpenFlow protocol. Second, it manages and monitors the computation nodes. It can launch or resize virtual instances, and it periodically monitors every computation node to check if there is any unexpected outage. Third, it serves as an agent of the ISP-wide resource scheduler. It updates the latest resource status to the scheduler, and receives the resource reservation requests from the scheduler.

4.1.2 Computation Nodes

We use virtualization to allow one computation node to host multiple running services at the same time. Among different virtualization technologies, we choose operating

system-level virtualization to host multiple isolated user-space instances. Compared with whole-system virtualization (e.g., VMWare vSphere) or paravirtualization (e.g., Xen), operating system-level virtualization has less overhead [20]. For our Linux-based prototype, we use LXC for virtualization.[1] Similar to public cloud services, every deployed service is contained in a LXC-based VI.

There are two physical network interfaces (NICs) in every computation node, i.e., a data NIC and a control NIC. The data NIC serves as the data delivery interface between the router and the computation node. The control NIC handles the management messages between the computation node and the cloudlet controller. We assume that the router selects x ports to devote to in-network services, and each of them can link the router to a computation node. If there are fewer allocated router ports than computation nodes, we use intermediate switches to connect the front-end router to the computation nodes.

4.1.3 Front-End Router

As the front-end router needs to handle both legacy traffic delivery and the demultiplexing of service-related traffic, we require this router to be compatible with the OpenFlow protocol. There are two main reasons for our choice. First, the OpenFlow protocol is easy to use. Demultiplexing policies can be established by configuring the "flow tables". Second, OpenFlow routers (or switches) are supported by mainstream routing hardware vendors, such as Cisco, Juniper, and HP. As a result, ISPs can choose among a variety of available models, and the same configuration script can be directly re-used in different networks.

The service providers might want to introduce some "complicated" traffic steering policies [21] that fall outside the scope of classic Layer 2 or Layer 3 functions that OpenFlow supports. Viable examples include service composition, load balancing among VIs, and dynamic traffic transformation. PacketCloud is compatible with SIMPLE [21], a SDN-based policy enforcement layer. With SIMPLE, different traffic steering policies can be translated into demultiplexing rules and can be installed into corresponding front-end routers.

4.2 Packet Demultiplexing and Processing in a VI

Since flow entries in OpenFlow are pro-actively installed before a service starts, a new incoming packet will be checked against matching fields of every flow entry, in an order of priority. Once it matches a service-related flow entry, it will be forwarded to the service. If it does not match any service-related flow entry, it will be delivered as a legacy packet according to its destination address instead of being redirected to the cloudlet.

PacketCloud services can run in different layers of the Internet protocol stack. For example, a web cache works at the application layer, and an IPSec tunnel works at the network layer. Irrespective of which layer a service focuses on, the service needs to deal with the traffic at the protocol data unit (PDU) level: at the network layer, a PDU is an IP packet; at the transport layer, a PDU is a segment (TCP) or a datagram (UDP); at the application layer, a PDU is a message. For services working at the application layer, a PDU might be composed of multiple IP packets due to the MTU constraint. In such a case, earlier packets of a PDU must be cached temporarily.

After a complete PDU is received by a service, there are three typical action types: (1) PDU dropping: an unwanted PDU will be dropped. A typical example of such action is a transparent IDS service. If a PDU is identified by IDS as a malicious PDU, it will be dropped without further delivery. (2) PDU forwarding: The service can either ship the received PDU back to the switch for further delivery without modification (e.g., PDUs passed the inspection of an IDS), or revise the PDU header and/or the PDU payload (e.g., an onion routing service which needs to both update the payload and the destination address). (3) Generating new PDUs: Sometimes the services need to generate additional PDUs. For example, once receiving a request to some content which has been cached, a web cache can send the cached content to the requestor directly, instead of forwarding the request to the original content provider.

TABLE 3
Evaluated PacketCloud Services

Service	Packet Processing Engine
Content cache	squid
Network IDS	snort
Anonymous communication	Tor
VPN gateway	OpenVPN
Load balancer	Balance
IPSec tunnel	OpenSwan
Traffic optimizer	WANProxy

4.3 Implemented and Evaluated Services

As shown in Table 3, we have implemented the following six services on commodity servers running Ubuntu 12.04. For each of them, we adopt an open source tool to build the packet processing engine. All these services have been tested in LXC-based virtual instances.

Visible services. We use OpenVPN[2] to establish a VPN gateway. An end user can use the deployed VPN gateway to access the Internet, and the traffic between them might be encrypted and/or compressed.

To support anonymous communication, we use Tor,[3] an open source implementation of onion routing. Different ISPs can deploy Tor nodes, and these nodes will form an in-network overlay to ensure the anonymity of end users' data traffic.

Transparent services. To implement a transparent content cache, we choose squid,[4] a widely used web cache proxy. The service provider needs to tell the front-end router what kind of packets it wants by specifying the source/destination addresses or port information.

To build rule-based intrusion detection system, we adopt snort,[5] an open source network intrusion prevention and detection system (IDS/IPS). As a transparent

1. http://lxc.sourceforge.net/

2. http://openvpn.net/
3. https://www.torproject.org/
4. http://www.squid-cache.org/
5. http://www.snort.org/

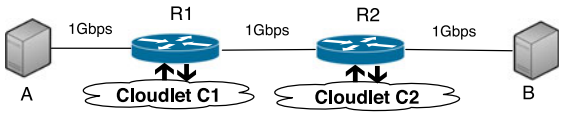


Fig. 4. Experiment topology on the DeterLab testbed.

service, a demultiplexing rule must be sent to the front-end router. Snort can be used to check the header and/or payload of intercepted network packets, and unwanted packets will be discarded.

We use balance[6] as a TCP proxy to achieve load balancing. A service provider can just announce the public IP of the load balancer to its customers, and this load balancer will distribute the incoming traffic among a number of backend servers.

To build an IPSec tunnel, we pick Openswan,[7] an IPsec implementation for Linux. An IPSec tunnel can secure the communications between two trustworthy branches. Openswan encrypts the traffic before leaving one branch, and decrypts the traffic after arriving at another branch.

5 Evaluation

In this section, we aim to systematically evaluate PacketCloud. In Section 5.1, using a fully functional emulation environment, we study the additional per-hop delay and scalability of PacketCloud services. In Section 5.2, we use Internet-based measurement to demonstrate why in-network cloudlets are useful to achieve a shorter "sender-service-receiver" latency. In Section 5.3, we introduce an Internet-based content delivery example, and show how PacketCloud can be beneficial for different Internet entities. In Section 5.4, we use IDS as another application to evaluate PacketCloud in both an emulated environment and the real Internet.

5.1 Evaluation in an Emulated Environment

We build a prototype of PacketCloud on the DeterLab testbed, a fully functional emulation environment.[8] To emulate an OpenFlow-based front-end router for a cloudlet, we utilize OpenvSwitch[9] to build a software-based OpenFlow-compatible router. Every computation node runs Ubuntu 12.04, and VIs are created using LXC.

As shown in Fig. 4, our experimental topology is composed of 10 DeterLab nodes. Each node has an Intel(R) Xeon (R) CPU E3-1260L running at 2.4 GHz. All data NICs and router ports works at a rate of 1 Gbps. There are two end hosts A and B, and two routers R_1 and R_2 stay in between. Since one DeterLab node used in our experiment has only five NICs, we can only attach at most three co-located computation nodes to a router. In our study, the router R_1 connects to three co-located nodes, i.e., C_{11}, C_{12}, and C_{13}. These three nodes form a simplified *cloudlet* C_1 with three computation nodes. Also, the router R_2 has a co-located cloudlet C_2, which is also composed of three computation nodes. Every link in this topology is a 1 Gbps link without any emulated delay or packet loss.

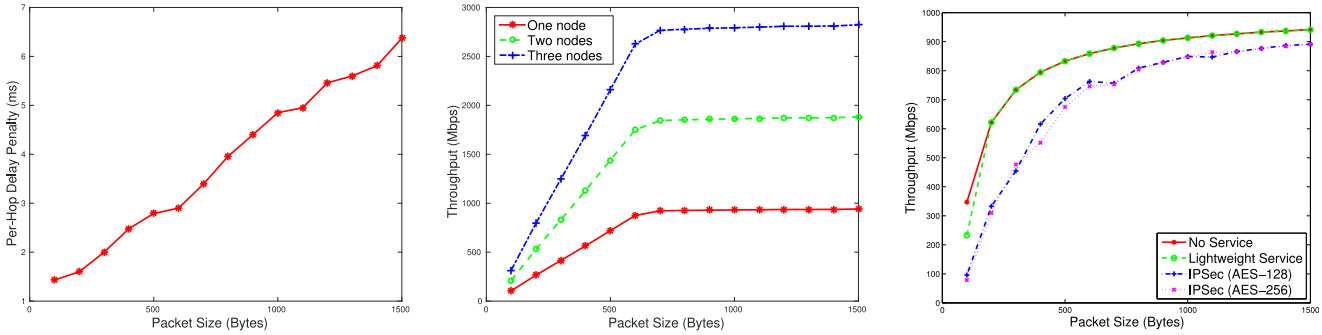
We conduct some performance-oriented microbenchmarking with two major focuses. First, *per-hop delay penalty*. In this section, we investigate a service running at the network layer. Therefore, we define the delay penalty as the duration between a router starts to forward a packet to its co-located cloudlet for processing, and the router receives the processed packet. We require this delay penalty to be small, as the end-to-end delay is critical for many Internet applications. Second, *scalability*. We need to ensure that a cloudlet is able to handle high-throughput data traffic.

As the first step, we focus on the cloudlet C_1 and its co-located router R_1, to see how much per-hop delay penalty it will introduce, and how much traffic it can deal with in a fixed time interval. We use a computationally intensive service, i.e., an AES encryption service (using AES-256-CBC mode), for our evaluation. We let the router R_1 forward fixed size packets to node C_{11} at a slow rate of one packet per second. By varying the packet size and repeating our experiment for 10 times, we can obtain the mean value of the per-hop delay penalty. According to Fig. 5a, we can see that when the packet size is 100 bytes, the delay penalty is only 1.43 ms. Even when the packet size becomes as large as 1,500 bytes (the default Ethernet MTU), the per-hop delay penalty is 6.37 ms. These values are smaller than the additional round-trip delay of using WiFi [22]. Therefore, even hosting a computationally intensive service, PacketCloud introduces a very small per-hop delay penalty for data packets, and the added delay is negligible for most existing Internet applications.

To evaluate the scalability, we use the maximum loss-free forwarding rate (MLFFR) metric [23], [24] for the evaluation. We let R_1 send fix-size packets to C_1 at a certain data rate, and the processed packets will be delivered back to R_1. We increase the input date rate gradually, and the MLFFR can be discovered as soon as the packet loss occurs. We repeat the experiment for 10 times, and record the average value of MLFFR. To see the impact of the cloudlet size, we vary the number of allocated computation nodes, i.e., investigating the use of one, two, and all three computation nodes, respectively. According to Fig. 5b, we can see that MLFFR grows linearly when we allocate more computation nodes to the cloudlet. Due to the restriction of the number of NICs in DeterLab, we are not able to add more computation nodes to the cloudlet. Nevertheless, in practice, commodity OpenFlow switches always have tens of ports, and accordingly we can host tens of computation nodes for the cloudlet. Therefore, if we have 11 such computation nodes for C_1, the cloudlet can handle more than 10 Gbps traffic for AES encryption. Meanwhile, when we decrease the packet size, the traffic throughput decreases accordingly. When we set the packet size as 100, we can still use 10 computation nodes to handle 1 Gbps traffic for AES encryption. These numbers can be further improved if we introduce better hardware, e.g., more advanced CPUs. Therefore, PacketCloud can scale well to high-throughput data traffic.

Furthermore, we evaluate PacketCloud from an end-to-end perspective, and we use the whole experimental topology shown in Fig. 4. We consider the following two scenarios. First, *default IP forwarding*. In this scenario, we do not involve any in-network service. The second scenario involves a

6. http://www.inlab.de/balance.pdf
7. https://www.openswan.org/
8. http://www.isi.deterlab.net/index.php
9. http://openvswitch.org/



(a) Per-hop delay penalty (b) MLFFR for the cloudlet (c) End-to-End Throughput

Fig. 5. Evaluation of PacketCloud on DeterLab.

computationally intensive service, i.e., an IPSec tunnel. We assume that the link $R_1 - R_2$ is untrustworthy. We randomly pick one node in C_1 and one node in C_2, and each of them runs an Openswan daemon. The node in C_1 serves as the ingress node of the IPSec tunnel, and the node in C_2 serves as the egress node of the IPSec tunnel. With the help of them, we can transfer encrypted traffic between R_1 and R_2. The complete path of the packet is $A \rightarrow R_1 \rightarrow C_1(encrypt) \rightarrow R_1 \rightarrow R_2 \rightarrow C_2(decrypt) \rightarrow R_2 \rightarrow B$. Similarly, for any data sending from B to A, it will be encrypted by C_2 and decrypted by C_1. We evaluate the IPSec tunnel from following two aspects. First, how much additional delay this tunnel will introduce. Second, to what extent the overall throughput will be degraded after introducing this tunnel. For the encryption algorithm in IPSec, we have evaluated two representative ones, i.e., AES-128-CBC, and AES-256-CBC.

We use the *ping* tool to measure the round-trip latency between A and B. We conduct the ping measurement for 10 independent times, and the average measured latency is 0.70 ms. For each of the two AES key size settings, the round-trip latency is about 1.69 ms. In other words, a 0.99 ms per-hop delay penalty is introduced. Such delay penalty is negligible for most existing Internet applications. For comparison, we also investigate the end-to-end performance after launching a lightweight service. We deploy a snort-based intrusion detection system on C_1 to inspect all data packets from A to B, and another snort-based intrusion detection system on C_2 to inspect all data packets from B to A. According to our measurement, the average round-trip time between A and B is 1.10 ms. In other words, such a lightweight service will introduce a 0.40 ms per-hop delay penalty.

As in [25], we use the *iperf* tool to measure the TCP throughput for data transfer from A to B. We run iperf for 10 independent times, and each time the measurement lasts 100 seconds. To see the impact of different packet sizes, we adjust the maximum transmission unit (MTU) of the links to set a series of upper bounds for the packet size. The results are shown in Fig. 5c. When we use the default setting in DeterLab (MTU = 1,500), we can see that the traffic throughput values are 942, 941, 891, and 891 Mbps, for default IP forwarding, a lightweight service, an IPSec service (AES-128-CBC), and an IPSec service (AES-256-CBC), respectively. When we choose a smaller packet size, the traffic throughput decreases accordingly. Nevertheless, we can still allocate more computation nodes to scale up the overall throughput.

5.2 Internet-Based Network Delay Evaluation

No matter a visible service or a transparent service, packets are always sent by a sender, and processed by a deployed service. If the service chooses not to drop the packets, the processed traffic will be delivered further to a receiver (the receiver can also be the sender itself in some scenarios). Therefore, we can use a "sender-service-receiver" pattern to represent typical PacketCloud traffic. we introduce an Internet-based measurement to compare different network locations for hosting in-network services, i.e., on default end-to-end paths, EC2 data centers, and in-network cloudlets. The metric we use in this section is *network delay* (d), i.e., the overall network data transmission delay of a "sender-service-receiver" path. We still use 580 PlanetLab nodes to serve as end users.

We consider the following three groups of locations to host in-network services:

- On-path: services are located on default end-to-end paths.
- Cloudlet: services are deployed on in-network cloudlets (As in Section 2, we use the same set of 32 randomly selected PlanetLab nodes).
- Amazon EC2: services deployed on Amazon EC2 data centers.

For the first scenario, we measure the end-to-end round-trip network delay by using ICMP ping, as the per-hop delay penalty is always less than 1 ms, which is negligible

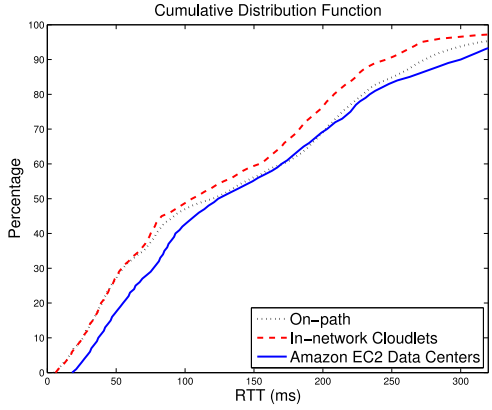


Fig. 6. Distribution of the network delay.

for most of the end-to-end paths among PlanetLab nodes. For the second and the third scenarios, we assume that the prospective service has been deployed at a group G of network locations. As we aim to seek for a shortest "sender-service-receiver" path, we can calculate the network delay d as $d = min_x(RTT(sender, x) + RTT(x, receiver)), x \in G$.

Fig. 6 shows the cumulative distribution function of the network delays. The median value of the three scenarios are 121, 101, and 130 ms, respectively. Among the three groups, deploying services on the EC2 data centers will result in largest network delay. Differently, deploying services on cloudlets can achieve an even lower network delay than using on-path services. We believe this is because suboptimal routing widely exists in today's Internet [26], [27], and using cloudlets will introduce some possible detour routes.

5.3 Application: Internet-Based Content Delivery

In this section, we use a content delivery example to demonstrate the usefulness of PacketCloud for different Internet entities. We pick this example because nowadays a large portion of today's Internet traffic is for content delivery, e.g., file sharing, and video delivery. For a popular content provider, it needs to handle a massive number of content requests at the same time. To save the incoming/outgoing bandwidth, content providers always aim to serve the end users in their local networks. The deployment of local caches is also attractive to ISPs and end users. For ISPs, they can also save their bandwidth, as a large portion of the redundant traffic could be eliminated. For end users, getting contents from local networks would be much faster than downloading them from the content provider's data center, which might be topologically far away.

As shown in Fig. 7, we build a simple topology based on seven PlanetLab nodes from three autonomous systems (ASes). The content provider is located in MIT (AS3). There are two end users. The first user is located in Beijing, China. This user connects to the content provider through an access router located in Shanghai, China, and this router has a co-located computation node in Shanghai as well. All these three nodes belong to China Education and Research Networks (CERNET, AS4538). The second user is from Berlin, Germany. This user connects to the content provider through an access router located in Berlin, and this router also has a co-located computation node in Berlin. All these three nodes belong to the German National Research and Education Network (DFN, AS680).

Among these seven nodes, the two access routers are programmable as they run OpenvSwitch. Each of them are

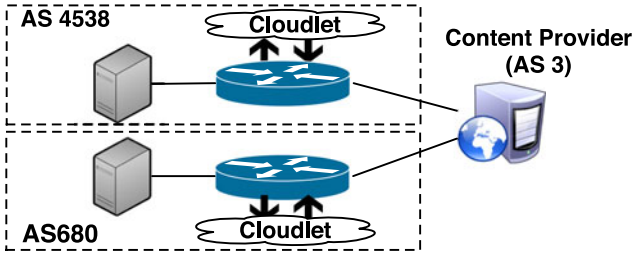


Fig. 7. Evaluation on Real Internet.

connected to the content provider by a virtual point-to-point link. Each client randomly generates HTTP requests for the server. Every request will pick among the 10,000 files available on the web server. As in [28], for every client, the popularity of these files follows a Zipf distribution. In our study, we set the Zipf parameter as 2. Every client generates 1,000 requests one after one.

Similar to last section, we study two scenarios, i.e., default IP forwarding, and transparent web cache. In the first scenario, the computation nodes do not participate. Each of the two clients exchanges data directly with the content provider through the corresponding router. In the second scenario, each of the two routers redirects the HTTP requests to content provider to the *squid* service running on the co-located computation node. We record the total incoming/outgoing traffic of the three ASes.

Our evaluation was conducted in on February 27, 2014. According to the results shown in Table 4, we can see that the use of PacketCloud can significantly reduce the incoming/outgoing traffic of the content provider. Also, the traffic between the content providers and the two end users' local ISPs can be largely saved as well.

For the benefits for end users, we compute the total downloading time in each scenario. For the user in China, its total downloading time in the first scenario and the second scenario are 1,888.04 and 885.15 seconds, respectively. For the user in Germany, its total downloading time in the first scenario and the second scenario are 1,104.76 and 409.51 seconds, respectively. Therefore, the use of PacketCloud can help users get content faster.

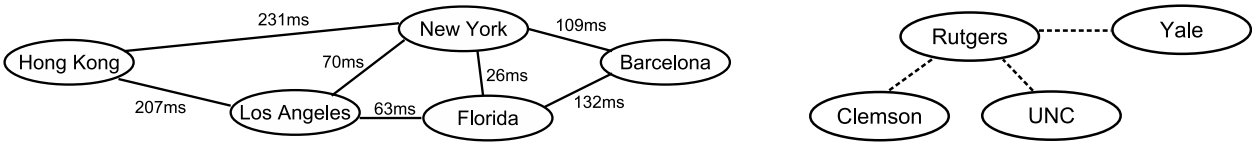
In short, PacketCloud providers a platform to host transparent web caches in strategic network locations flexibly. Different entities, including content providers, ISPs, and end users, can receive viable benefits from the wide deployment of PacketCloud.

5.4 Application: Intrusion Detection System

In this section, we use another application, i.e., an intrusion detection system, to demonstrate the usefulness of PacketCloud in handling malicious traffic. As shown in Fig. 8, we

TABLE 4
Incoming/Outgoing Traffic of the Content Provider and Eyeball ISPs

	Content Provider AS3		Eyeball ISP AS4538		Eyeball ISP AS680	
	In	Out	In	Out	In	Out
w/ Cache	3.2 MB	96.2 MB	40.3 MB	1.3 MB	55.9 MB	1.9 MB
w/o Cache	9.3 MB	348.8 MB	171.4 MB	4.3 MB	177.4 MB	5.0 MB



(a) DeterLab (SWAN Topology) (b) PlanetLab

Fig. 8. Experimental topology for IDS.

use two topologies in our study. One is an emulated topology on DeterLab, the other one is a virtual topology built on top of the real Internet.

The DeterLab topology in our study has been used in [29]. There are five nodes (Hong Kong, Los Angeles, New York, Florida, Barcelona) and seven links in this topology. This topology presents a global software-driven WAN. As in [29], the RTTs among these nodes are emulated based on geographic distances, and detailed values are shown in Fig. 8. For each link, the capacity is set as 1 Gbps, and the packet loss rate is set as 0.01 percent. For simplicity, each node has one co-located computation node. We assume that the user A accesses the Internet via the Hong Kong node, and the user B accesses the Internet via the Barcelona node. Also, there are two attackers, and they get their connectivity via the New York node, and the Florida node, respectively. Each of these attackers will use the *hping3* tool to generate a ping flood by sending ICMP ping packets to user A. In our study, each attacker sends out 10,000 packets per second, and the size of each packet is 1,000 bytes. In our evaluation, we use iperf to measure the end-to-end throughput between A and B. We let user A run an iperf server, and let user B run an iperf client. If we do not enable any in-network IDS, we can see that the average end-to-end throughput is 1.27 Mbps for 10 independent measurements. Differently, we can use PacketCloud to deploy the in-network IDS service on the computation nodes co-located with the New York node and the Florida node, and the service is configured to filter these ping flood packets. As soon as these firewalls are enabled, the average end-to-end throughput will become 9.51 Mbps for 10 independent measurements, which is about 7.5 times larger.

The virtual topology built on the real Internet using four PlanetLab nodes. These nodes are from Yale University, Rutgers University, Clemson University, University of North Carolina at Chapel Hill (UNC). As a result, we can do experiments on a network with realistic background traffic. As shown in Fig. 8b, these nodes are connected by some virtual point-to-point links. These nodes serve as the backbone of a virtual network, and each of them has a co-located PlanetLab node serving as the computation node. We use another Rutgers node to host an iperf server, and we call it "Rutgers user". Also, we adopt another Yale node to host an iperf client, and we call it "Yale user". The two attackers are from Clemson and UNC, respectively. We executed 10 independent experiments at around 1 PM on December 30, 2014 (Eastern Standard Time). To emulated malicious traffic, we let each attacker send 1,000 ICMP ping packets per second to the Rutgers user. In each round of experiments, we use iperf to measure the end-to-end throughput between the Yale user and the Rutgers user, and the average measured throughput is 3.84 Mbps. Differently, when we have enabled the in-network IDS service on the computation nodes at Clemson and UNC, the average measured end-to-end throughput between the Yale user and the Rutgers user becomes 31.6 Mbps.

Therefore, in both an emulated environment and the real Internet, PacketCloud can help introduce an in-network IDS service, which is very helpful to filter the malicious packets out.

6 Discussion

After demonstrating the design, implementation, and evaluation of PacketCloud, we provide a series of discussions for other key issues.

6.1 Underlying Network Architecture

PacketCloud is compatible with different underlying network architectures. We can either use conventional IP, or clean-slate architectures. In this paper, we demonstrate the usefulness of building PacketCloud on top of IP. We believe that PacketCloud would be very helpful for a variety of network architectures, especially for emerging future Internet architectures like MobilityFirst [30], and RBF [31]. In an earlier publication of PacketCloud [32], we have demonstrated how to make PacketCloud as a key component of the MobilityFirst architecture, which is designed to help the Internet better support emerging mobile services and applications.

6.2 Incentive for Different Internet Entities

The Internet is a huge and complicated system, and PacketCloud aims to provide a viable incentive to different Internet entities. In this section, we describe why PacketCloud is helpful for end users, third-party providers, and ISPs.

End users. PacketCloud can help end users from various aspects. For example, the users can get a higher throughput for content fetching, or can deliver their data in a confidential way, or can save the energy of their mobile devices. The end users can flexibly pick any deployed service to enhance their user experience.

Third-party providers. PacketCloud can help third-party providers get their services close to end users. The flexibility of cloudlet deployment allows third-party providers to deploy their instances deeply into different networks. Such deployment can not only satisfy the end users, but also help third-party providers reduce the incoming traffic for their dedicated data centers. For third-party providers, they do not need to deliver physical machines like Netflix's Open-connect, and they can save the front-end investment.

ISPs. ISPs need to make investment for deploying the cloudlets. Fortunately, PacketCloud is economically sound. First, PacketCloud can help ISPs tailor the network traffic. By eliminating redundant/malicious traffic, the network infrastructure can allocate more resources to serve legitimate users efficiently. Second, ISPs can lease

available cloudlets to third-party providers, and get rewards accordingly.

6.3 Prospective Future Directions

Now we have a proof-of-concept implementation of PacketCloud, and an experimental deployment of PacketCloud on the PlanetLab platform. In the next step, we plan to tackle some practical challenges for deploying PacketCloud in a larger scale. We are collaborating with some ISPs in mainland China, for example, the Oriental Cable Network (OCN) in Shanghai, to deploy PacketCloud in their infrastructure. OCN is the largest ISP in Shanghai, and we aim to use PacketCloud to host a number of scalable and extensible in-network services for OCN users. This will be an important part of OCN's plan to enhance its user experience.

Another prospective future topic is building a distributed version of the ISP-wide resource scheduler. We aim to build a platform similar to Onix [33], which allows multiple servers to work together as a cluster of integrated resource scheduler. This platform will efficiently exchange information among the servers, and will allow ISPs to add/remove servers to this cluster on demand. Compared with a single server architecture, such a distributed architecture can serve more service providers simultaneously, and is more robust to malicious attacks.

7 Related Work

There are a few existing proposals for in-network services. To host in-network services in the application layer, overlay networks are proposed, and we discuss overlay-based in-network services in Section 7.1. To handle high throughput data traffic, we investigate the middlebox consolidation technology in Section 7.2. Content delivery networks (CDNs) as a widely used in-network content caching service, is studied in Section 7.3. Active networks is discussed in Section 7.4.

7.1 Overlay-Based In-Network Services

Overlay networks, such as Internet Indirection Infrastructure (i3) [34], aim to accommodate in-network services in the application layer. Such solutions do not need to make any substantial change in the Internet infrastructure. However, as the service hosting nodes are not located on the data forwarding paths, data packets have to detour to utilize the services. Moreover, the capacity of a single overlay node is always limited. Therefore, handling high throughput data traffic would be difficult.

7.2 Middlebox Consolidation

CoMb [8] proposed the idea of middlebox consolidation. This solution can efficiently utilize the available resources for service hosting. However, CoMb is designed for enterprise networks, thus there is an assumption that all deployed functions are legitimate. It needs thorough knowledge of deployed applications for performance optimization, and is not open to emerging third-party content/application providers. In contrast, PacketCloud is an open platform to host services from different service providers, the deployed services could be experimental, or even malicious. Virtualization is necessary to provide isolation among them. We have demonstrated how to use virtualization to ensure the reliability and security of PacketCloud in Section 3.6.

ClickOS [7] proposed a Xen-based software platform, which allows one node to host multiple virtual machines to support in-network services. However, ClickOS requires all services to be implemented using Click modular router [35], and it cannot support general-purpose services. Moreover, even some Click-based services cannot be hosted by ClickOS directly, as tens of Click elements related to the file-system are not supported at the moment.

7.3 Content Delivery Networks

Nowadays a large portion of the Internet data are delivered through content delivery networks like Akamai and Amazon CloudFront. The goal of CDNs is to deliver content to end users with a high throughput and low latency. PacketCloud can provide more flexibility than CDNs for third-party providers. First, PacketCloud can host general-purpose services instead of being customized for content delivery. Second, even for the content delivery scenario, a CDN-based solution requires negotiations between the ISPs, the content providers, and the CDN companies. Instead, the use of PacketCloud simplifies the negotiations as the content providers can directly contact corresponding ISPs, instead of involving a CDN company. Particularly, for the ISPs which are not covered by mainstream CDNs, they can directly launch cloudlets in their networks, and collaborate with the third-party providers for in-network services like content caching.

7.4 Active Networks

Active Networks [36] proposed the idea of performing customized computations in the network. PacketCloud shares this vision, while satisfies a number of practical requirements such as efficient packet demultiplexing for high-throughput traffic, elastic packet processing, and security. Compared with NetServ [24], an integrated active network system for helping content providers, PacketCloud has several advantages. For instance, PacketCloud supports visible services, and is open to third-party providers. Also, the service modules in NetServ are required to be implemented by Java, which is not efficient.

8 Conclusion

This paper proposes PacketCloud, an architectural solution to host elastic services in the network infrastructure. As an open platform, PacketCloud helps both ISPs and third-party providers deploy in-network services conveniently and incrementally. Our evaluation demonstrates the usefulness of PacketCloud. PacketCloud has a small delay penalty, and can scale well for high-throughput data traffic. Different from public data centers, PacketCloud can get in-network services closer to end users, and provide viable incentives for different Internet entities.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 1040043, and an AWS in Education Research Grant award. The authors thank

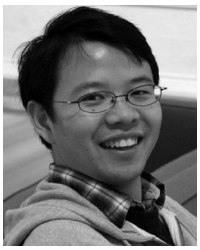
Feixiong Zhang, Dr. Kiran Nagaraja, and Prof. Dipankar Raychaudhuri from Rutgers University for their comments. The preliminary version of this paper was published in [32]. Yang Chen is the corresponding author.

REFERENCES

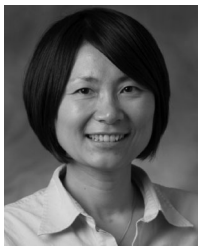
[1] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Trans. Comput. Syst.*, vol. 2, no. 4, pp. 277–288, 1984.
[2] T. Vu, A. Baid, Y. Zhang, T. D. Nguyen, J. Fukuyama, R. P. Martin, and D. Raychaudhuri, "DMap: A shared hosting scheme for dynamic identifier to locator mappings in the global internet," in *Proc. IEEE 32nd Int. Conf. Distrib. Comput. Syst.*, 2012, pp. 698–707.
[3] F. Dogar, A. Phanishayee, H. Pucha, O. Ruwase, and D. Andersen, "Ditto: A system for opportunistic caching in multi-hop wireless mesh networks," in *Proc. ACM 14th ACM Int. Conf. Mobile Comput. Netw.*, 2008, pp. 279–290.
[4] M. Li, D. Agrawal, D. Ganesan, and A. Venkataramani, "Block-switched networks: A new paradigm for wireless transport," in *Proc. 6th USENIX Symp. Netw. Syst. Des. Implementation*, 2009, pp. 423–436.
[5] X. Liu, X. Yang, and Y. Lu, "To filter or to authorize: Network-layer dos defense against multimillion-node botnets," in *Proc. ACM SIGCOMM*, 2008, pp. 195–206.
[6] B. Carpenter and S. Brim, "Middleboxes: Taxonomy and issues," RFC 3234, 2002.
[7] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "ClickOS and the art of network function virtualization," in *Proc. 11th USENIX Conf. Netw. Syst. Des. Implementation*, 2014, pp. 459–473.
[8] V. Sekar, N. Egi, S. Ratnasamy, M. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Proc. 9th USENIX Conf. Netw. Syst. Des. Implementation*, 2012, p. 24.
[9] Netflix open connect content delivery network. [Online]. Available: https://signup.netflix.com/openconnect, 2013.
[10] 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 *Proc. ACM Conf. Appl., Technol., Archit. Protocols Comput. Commun.*, 2012, pp. 13–24.
[11] M. Satyanarayanan, P. Bahl, R. Cáceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct.-Dec. 2009.
[12] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: Comparing public cloud providers," in *Proc. 10th ACM SIGCOMM Conf. Internet Meas.*, 2010, pp. 1–14.
[13] N. Laoutaris and P. Rodriguez, "Good things come to those who (Can) wait," in *Proc. ACM HotNets*, 2008, pp. 1–6.
[14] M. Sherr, A. Mao, W. R. Marczak, W. Zhou, B. T. Loo, and M. Blaze, "A3: An extensible platform for application-aware anonymity," in *Proc. 17th Annu. Netw. Distrib. Syst. Security Symp.*, 2010, pp. 1–20.
[15] M. Edman and P. Syverson, "AS-awareness in tor path selection," in *Proc. 16th ACM Conf. Comput. Commun. Security*, 2009, pp. 380–389.
[16] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proc. 4th USENIX Conf. Netw. Syst. Des. Implementation*, 2007, p. 17.
[17] A. Li, X. Liu, and X. Yang, "Bootstrapping accountability in the internet we have," in *Proc. 8th USENIX Conf. Netw. Syst. Des. Implementation*, 2011, pp. 155–168.
[18] A. Shieh, S. Kandula, and E. G. Sirer, "SideCar: Building programmable datacenter networks without programmable switches," in *Proc. 9th ACM SIGCOMM Workshop Hot Topics Netw.*, 2010, pp. 21:1–21:6.
[19] V. Valancius, C. Lumezanu, N. Feamster, R. Johari, and V. V. Vazirani, "How many tiers? pricing in the internet transit market," in *Proc. ACM SIGCOMM*, 2011, pp. 194–205.
[20] J. Whiteaker, F. Schneider, R. Teixeira, C. Diot, A. Soule, F. Picconi, and M. May, "Expanding home services with advanced gateways," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 5, pp. 37–43, 2012.
[21] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "SIMPLE-fying middlebox policy enforcement using SDN," in *Proc. ACM SIGCOMM*, 2013, pp. 27–38.
[22] E. Cuervo, A. Balasubramanian, D.-K. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making smartphones last longer with code offload," in *Proc. ACM 8th Int. Conf. Mobile Syst., Appl. Serv.*, 2010, pp. 49–62.
[23] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, 2000.
[24] S. Srinivasan, J. W. Lee, E. Liu, M. Kester, H. Schulzrinne, V. Hilt, S. Seetharaman, and A. Khan, "NetServ: Dynamically deploying in-network services," in *Proc. ACM ReArch*, 2009, pp. 37–42.
[25] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In VINI veritas: Realistic and controlled network experimentation," in *Proc. ACM Conf. Appl., Technol., Archit. Protocols Comput. Commun.*, 2006, pp. 3–14.
[26] C. Lumezanu, R. Baden, N. Spring, and B. Bhattacharjee, "Triangle inequality variations in the internet," in *Proc. 9th ACM SIGCOMM Conf. Internet Meas. Conf.*, 2009, pp. 177–183.
[27] S. Lee, Z.-L. Zhang, S. Sahu, and D. Saha, "On suitability of euclidean embedding for host-based network coordinate systems," *IEEE/ACM Trans. Netw.*, vol. 18, no. 1, pp. 27–40, Feb. 2010.
[28] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K. Ng, V. Sekar, and S. Shenker, "Less pain, most of the gain: Incrementally deployable ICN," in *Proc. ACM SIGCOMM*, 2013, pp. 147–158.
[29] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven WAN," in *Proc. ACM SIGCOMM*, 2013, pp. 15–26.
[30] D. Raychaudhuri, K. Nagaraja, and A. Venkataramani, "MobilityFirst: A robust and trustworthy mobility-centric architecture for the future internet," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 16, no. 3, pp. 2–13, 2012.
[31] L. Popa, N. Egi, S. Ratnasamy, and I. Stoica, "Building extensible networks with rule-based forwarding," in *Proc. 9th USENIX Conf. Operating Syst. Des. Implementation*, 2010, pp. 1–14.
[32] Y. Chen, B. Liu, Y. Chen, A. Li, X. Yang, and J. Bi, "PacketCloud: An open platform for elastic in-network services," in *Proc. 8th ACM Workshop Mobility Evolving Internet Archit.*, 2013, pp. 17–22.
[33] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A distributed control platform for large-scale production networks," in *Proc. 9th USENIX Conf. Operating Syst. Des. Implementation*, 2010, pp. 1–14.
[34] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," *IEEE/ACM Trans. Netw.*, vol. 12, no. 2, pp. 205–218, Apr. 2004.
[35] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek, "The click modular router," *SIGOPS Oper. Syst. Rev.*, vol. 33, no. 5, pp. 217–231, 1999.
[36] D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall, and G. Minden, "A survey of active network research," *IEEE Commun. Mag.*, vol. 35, no. 1, pp. 80–86, Jan. 1997.

Yang Chen received the BS and PhD degrees from the Department of Electronic Engineering, Tsinghua University, in 2004 and 2009, respectively. His PhD advisor is Prof. Xing Li. He is a pre-tenure associate professor within the School of Computer Science at Fudan University. Before that he was a postdoctoral associate within the Department of Computer Science at Duke University from April 2011 to September 2014. His research interests include Internet architecture, online/mobile social networking, and cloud computing. He is a senior member of the IEEE.

Yu Chen received the BS and MS degrees in computer science from the National University of Singapore, Singapore and Duke University, in 2010 and 2013, respectively. His research interests include computer networks and security.



Qiang Cao is a Postdoctoral Associate in the department of computer science at Duke University. His research interests are in networked systems and security. He received a PhD in computer science from the same university in 2014.



Xiaowei Yang received the BE degree in electronic engineering from Tsinghua University in 1996, and the PhD degree in computer science from MIT in 2004. She is an associate professor of computer science at Duke University. Her research area is in networks and distributed systems, with an emphasis on architecture design and security.