Support for Service Composition in $i3$ *

Karthik Lakshminarayanan Ion Stoica Klaus Wehrle
University of California, Berkeley Univ. of Tübingen

ABSTRACT

We consider the problem of service composition in a wide area network, where an end-user can send its packets through intermediate processing points (middleboxes) which can perform a variety of services. Example of such services are filtering, intrusion detection, anonymization, transcoding, and caching. In this paper, we argue that the Internet Indirection Infrastructure ($i3$)—an overlay network architecture that enables users to locate services and control the path followed by their packets—provides a natural platform for service composition. We discuss the challenges in implementing service compositions on top of $i3$, and suggest several approaches to address these challenges.

Categories & Subject Descriptors: H.4.3 [**Information Systems Applications**]: Communication Applications

General Terms: Design

Keywords: Service Composition, Internet, Indirection

1. INTRODUCTION

There are two major approaches of deploying services in today's Internet. The first approach is to deploy services at the access points to the Internet. In this case, all hosts behind an access point can benefit from the service running at the access point. Example of such services are intrusion detection, filtering, and caching. A second approach is to deploy services at various locations through the Internet, and then use DNS or HTTP to redirect the packets to a machine implementing the desired service. However, neither of these approaches is general or flexible enough to support arbitrary service composition. The first approach is limited to hosts behind the access point, while the second approach is typically limited to server selection, as it does not allow receivers to specify intermediate service points.

To achieve full service composability, one would need the ability to find the desired services, locate machines that run these services, and explicitly forward the traffic through these machines. While this functionality can be implemented in one monolithic system, an arguably better approach would be to isolate the common primitives required for providing service composition, and embed them in a shared infrastructure. This approach would lower the barrier of deployment for new services in the Internet.

In this paper, we consider such a shared infrastructure, called Internet Indirection Infrastructure $i3$ [10], and show how it can be used to enable service composition. $i3$ is a routing infrastructure in which hosts can set-up paths or use source routing to forward their packets. A hop in $i3$ can represent a service. In addition, $i3$ provides support for multicast, anycast and mobility. We discuss how applications can use these primitives to locate nearby or lightly loaded machines running a given service, and to recover from failures.

*Karthik Lakshminarayanan and Ion Stoica are supported by NSF under Grants ANI-0133811 and ANI-0085879. Klaus Wehrle is supported by Baden-Württemberg (Germany) state foundation.

ACM Multimedia '04, October 10-16, New York, NY USA

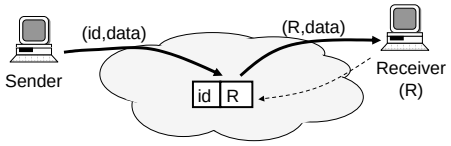

Figure 1: Sending a packet with ID id to receiver R in $i3$.

2. $i3$ OVERVIEW

In this section, we provide a brief overview of $i3$ [10]. In a nutshell, $i3$ provides indirection, that is, it decouples the act of sending a packet from the act of receiving it. There are two basic operations in $i3$: sources send packets to a logical *identifier* (ID) and receivers express interest in packets by inserting a *trigger* into the network (Figure 1(a)). Triggers can be thought of as routing entries that point to receivers or to other triggers. Packets are of the form $(id, data)$ and triggers are of the form $(id, addr)$, where $addr$ is either an ID or an IP address. Given a packet $(id, data)$, $i3$ finds the trigger $(id, addr)$, and forwards $data$ to $addr$. Triggers are maintained using soft-state. Receivers refresh their triggers for as long as they desire to receive packets sent to those triggers.

Identifiers in $i3$ are 256 bits long. IDs in packets are matched with those in triggers using longest prefix matching, where the prefixes are at least 192 bits. $i3$ is implemented as an overlay network of nodes that store triggers and forward packets. Identifiers are mapped to $i3$ nodes using a distributed lookup service such as Chord [11]. A trigger is stored at the node that is responsible for its identifier in accordance to the lookup service. Similarly, packets are routed to the appropriate node using the lookup service.

$i3$ provides the following communication primitives:

Mobility. A mobile host that changes its address from R to R' can preserve the end-to-end connectivity by updating its trigger from (id, R) to (id, R').

Multicast. Creating a multicast group is equivalent to having all members of the group register triggers with the same ID.

Anycast. All hosts in an anycast group maintain triggers that have identical 192-bit prefixes, but different 64-bit suffixes. Packets are delivered to the group member that has the trigger with the longest matching identifier.

In addition, $i3$ allows either the sender or the receiver to forward packets through intermediate points in the network. One way of performing this in $i3$ is by replacing the packet ID with a stack of IDs. Forwarding such a packet is similar to source routing in IP. Similarly, a receiver can control packet

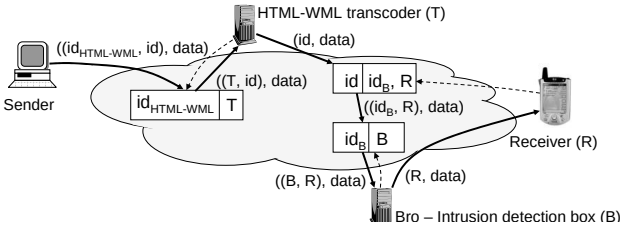

Figure 2: Sender and receiver control in $i3$.

forwarding by replacing the second field of its trigger with a stack that describes the forwarding path. Controlling packet forwarding at the ID level represents the key feature that enables service composition in $i3$.

In general, consider a packet $((id_1, id_2, \ldots, id_k), data)$ and a trigger $(id, (id'_1, id'_2, \ldots, id'_m))$. The packet matches the trigger if the first ID in the packet's stack id_1 matches the trigger's ID, id. If they match, the first ID in the stack is replaced by the trigger's stack, and the resulting packet, $((id'_1, \ldots, id'_m, id_2, \ldots, id_k), data)$, is forwarded to id'_1.

2.1 Performance Optimizations

Caching. To improve performance, $i3$ performs aggressive caching. In Figure 1, when packet $(id, data)$ first reaches the $i3$ node storing trigger (id, R) (call that node M) it caches M, and send all subsequent packets directly via IP to M.

Nearby triggers. While caching makes sure that most packets are routed through only one intermediate $i3$ node, that node can be far away from both the sender and the receiver. To alleviate this problem, receivers can sample the ID space, and choose a trigger with an ID that maps on a nearby $i3$ node.[1]

Shortcuts. Using the above optimizations, every packet is still relayed through at least one $i3$ node. While this indirection is necessary for useful functionalities such as multicast and mobility, it is, in general, less efficient than direct IP communication. To address this problem, we allow the communication path between an sender and receiver to circumvent $i3$. In particular, instead of caching the $i3$ node storing the receiver's trigger, a client is allowed to cache the receiver's address itself. The subsequent packets are sent then from source to destination via IP. Note that in this case the $i3$ is used as a lookup infrastructure (that resolves an ID to an IP address) instead of a forwarding infrastructure. Shortcuts are used only if both the sender and the receiver allow its use.

3. EXAMPLES

In this section, we give two examples of how one can use $i3$ to provide service composition.

Figure 2 illustrates the ability of both the sender and the receiver to forward packets through intermediate nodes (middleboxes) that implement third-party services. The sender sends an HTML web page to a PDA identified by ID id. The PDA runs WML, a lightweight version of HTML designed to run on wireless devices with limited capabilities.

The sender uses a stack of IDs $(id_{HTML-WML}, id)$ to forward the packets, first to the middlebox T (which performs HTML-WML transcoding), and then to the receiver. When the packet matches trigger $(id_{HTML-WML}, T)$, the first ID of the packet's stack $(id_{HTML-WML})$ is replaced with the second field of the trigger (T), and the resulting packet is forwarded to T. Upon receiving this packet, T performs transcoding, removes the first ID from the stack, and forwards the resulting packet $(id, data)$.

In turn, receiver R uses $i3$ to forward all packets it receives at ID id to an intrusion detection box such as Bro [9]. To achieve this, the receiver inserts a trigger $(id, (id_B, R))$, where id_B identifies the intrusion detection middlebox. Upon packet $(id, data)$ matching the trigger, the packet's ID is replaced with the trigger's second field. The resulting packet, $((id_B, R), data)$ is forwarded, via trigger (id_B, B), to node B and then finally to receiver R.

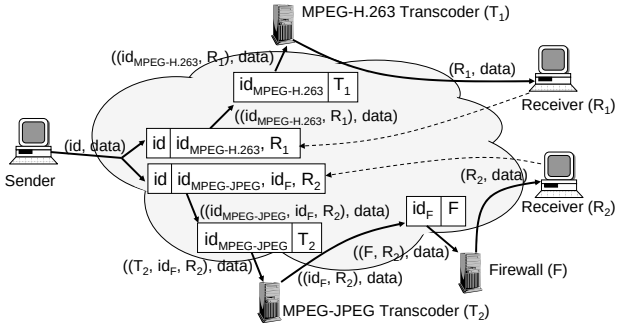

Figure 3: Heterogeneous multicast in $i3$.

The example in Figure 3 shows how $i3$ can be used to implement a heterogeneous multicast solution where different receivers have different needs. While the sender sends a video stream in MPEG format, receiver R_1 wishes to receive data in H.263 format, and receiver R_2 wishes to receive data in JPEG format. In addition, receiver R_2 wishes that all its incoming traffic to traverse a firewall F. Figure 3 illustrates how the sender and the receivers use $i3$ to achieve their goals. The sender simply sends packets of the form $(id, data)$. In turn, receiver R_1 maintains a trigger $(id, (id_{MPEG-H.263}, R_1))$, while R_2 maintains a trigger $(id, (id_{MPEG-JPEG}, id_F, R_2))$.

4. DESIGN CHALLENGES

While the examples presented in Section 3 illustrate the flexibility of $i3$ which allows both senders and receivers to compose services by routing packets to arbitrary IDs, there are three key issues that remain to be answered: (1) how do end-hosts determine the logical services to be interposed between the sender and receiver, (2) how do end-hosts locate middleboxes that satisfy their end-to-end performance requirements, and (3) how can a connection survive a middlebox failure. Since the first issue is largely orthogonal to $i3$—$i3$ can leverage any of the previously proposed solutions (such as [12]) to compute the service work-flow—in the remainder of this section we focus on the last two issues. We present these two issues in more detail, and make several suggestions on how to address them.

4.1 Path Selection

Consider a set of middleboxes that host a set of services, where each middlebox can host more than one service. These middleboxes form an overlay network, where the middleboxes can communicate with each other. Now, consider an end-host that wants to send its packets through a sequence of services $s_1, s_2, \ldots, s_k$. The question is then: how does the end-host select which middleboxes to use? Usually, such selection is subject to end-host performance constraints like delay and

[1]Such triggers are called private triggers. In addition to private triggers, a node (server) that wants to be reachable maintains a public trigger, *i.e.*, a trigger with a well known ID. Once a client contacts a server through its public trigger, they can exchange a pair of IDs which they use for the remainder of the communication.

bandwidth. This problem can be formulated as a constrained routing problem, which has been shown to be NP-hard [4]. Rather than concentrating on the algorithmic aspects of this problem, we discuss how we can leverage previously proposed approximation algorithms [1, 3, 12] in the context of $i3$.

Centralized (Oracle) Approach. One simple approach is to have an "Oracle" that has information about the resources availability at each middlebox, and the characteristics of the paths between these middleboxes. This requires each middlebox to periodically monitor and report to the Oracle its available resources, and the characteristics of the paths to other middleboxes. When a host wants to communicate to another host it simply queries the Oracle for a path that satisfies its performance requirements.[2] The Oracle consults its database, and, if it finds such a path, it replies to the host. Upon receiving the reply, the host uses $i3$ to forward the packets along the path.

The advantage of this approach lies in its simplicity and flexibility. Since the Oracle has full information about the overlay network formed by the middleboxes, it can use any of the previously developed algorithms to find a path that meets the performance requirements of the host.

On the downside, this approach is neither scalable nor robust, as all queries are handled by the Oracle. The natural solution to address these problems is to replicate the Oracle functionality. In particular, the Oracle can be implemented using a set of K servers, where each of these servers stores the information about the entire overlay network of middleboxes. To efficiently disseminate the measurement information from servers to the Oracle servers, we can use the $i3$ multicast primitive [6]. Finally, to reach one of the Oracle servers, hosts can use $i3$ anycast.

The assumption behind this solution is that the bottleneck is the query load, and not the measurement overhead, or the overhead of updating the Oracle servers. This is arguably a reasonable assumption if middleboxes are located at no more than several tens of locations in the Internet, as the middleboxes at the same location can share the network measurements. Let N be the number of middleboxes and let L be the number of locations. Then each Oracle server needs to maintain $O(N + L^2)$ state, and the average network measurement information generated by a location is $O(L)$.

If, on the other hand, middleboxes are deployed at hundreds or thousands of locations across the Internet[3], we need more sophisticated solutions that partition the database across multiple servers. The challenge here is in partitioning the overlay network without compromising the quality of path selection at the same time minimizing the average number of Oracle servers that are involved in selecting a path.

One general problem with the Oracle approach is that the Oracle knows only the characteristics of the paths between middleboxes, and not of the paths from end-hosts to middleboxes. Thus, the performance characteristics of these paths need to be estimated by end-hosts themselves. While some of the characteristics such as delay and loss rate can be estimated in a scalable fashion (using systems like GNP [8] and tomography techniques [2]), how to scalably estimate certain other characteristics such as available bandwidth remains an open problem.

Distributed Approach. Next, we consider an alternate approach that leverages the anycast functionality provided by $i3$, and requires no additional infrastructure. While this solution is less general than previously proposed distributed algorithms to perform constrained based routing, it is simpler and easier to implement.

[2]Note that the Oracle can also compute the service work-flow based on the sender and receiver capabilities and requirements.

[3]This scenario is consistent with the Akamai deployment model.

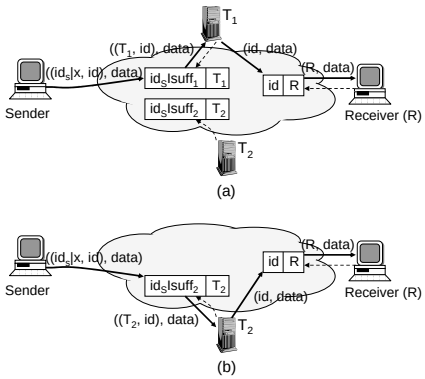

Figure 4: Using anycast for transparent recovery. (a) Middleboxes T_1 and T_2 provide the same service S and subscribe to the same anycast ID, $(id_s|*)$. Packets sent by the sender match the trigger of T_1. (b) After T_1 fails, its trigger times out, and the packets are routed to T_2.

Recall that $i3$ uses the longest prefix matching rule to match the packet's ID to the trigger's ID. An application can take advantage of this matching rule to select middleboxes according to different performance criteria. The idea is to encode applications semantics in the ID suffixes of both trigger and packet IDs.

For example, to ensure that a host selects a nearby middlebox, we can have each middlebox encode its location (*e.g.*, zip code) into the suffixes of its trigger IDs, and each host encode its location into the suffixes of its packet IDs [10]. Similarly, to ensure load balancing, each middlebox can adaptively vary the number of triggers that it maintains for each service as a function of its load. Finally, it is possible to implement a selection policy that takes into account both the load and the location of a middlebox [7]. The idea is to use GNP to assign to each middlebox a location in a Cartesian coordinate space, and then have the middlebox maintain a responsible region (in the Cartesian space) around this location. The packets of a host are forwarded to a middlebox whose region covers the host's location in the coordinate space. The middlebox can then vary its region as a function of its load, by inserting or removing triggers with appropriate suffixes [7].

When the path involves more than one service, each middlebox can recursively use the anycast functionality to pick a middlebox running the next service along the path. While such a greedy solution is easy to implement and does not require any additional infrastructure, in general, it cannot provide paths as good as the centralized solution can. One challenge is that it is unclear how to use $i3$ anycast to optimize for metrics other than latency (such as throughput and loss rate). Another challenge is that different hosts/middleboxes can simultaneously and independently select the same middleboxes, which can lead to sub-optimal selection.

4.2 Failure Recovery

Anycast provides natural support to recover from middlebox failures. If all middleboxes running a service subscribe to the same anycast group, then when one middlebox fails, the traffic is switched to another middlebox in the same group (see Figure 4). However, this represents only half of the solution. In cases where the middlebox maintains per connection state, just switching the traffic to another middlebox upon failure is not sufficient as the original state is lost.

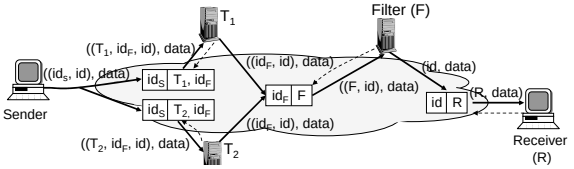

Figure 5: Send each packet to two middleboxes T_1 and T_2 that provide the same service. Middlebox F filters out the duplicate packets. If T_1 or T_2 fails, the other middlebox will continue to process and forward the packets to R, completely transparent to both the sender and receiver.

In the remainder of this section, we discuss three approaches to address this problem. These approaches differ in the amount of additional resources they require and the degree of transparency to end-hosts.

Shared Backup. In this approach, a middlebox periodically saves the state associated with a connection to a shared storage infrastructure such as OpenHash [5]. When the traffic of a connection switches to a backup middlebox, the middlebox retrieves the latest connection state from the shared storage infrastructure. There is a clear trade-off between the frequency of saving the connection state and the accuracy of restoring the state in case of failure. While the overhead can be reduced by saving only the incremental changes of the connections state, maintaining the exact state requires, at the limit, saving the state after receiving every packet, which is clearly infeasible.

Backup Middlebox. In this approach, each packet of the connection is replicated and sent to a backup middlebox. With $i3$ this can be easily achieved by having the middleboxes subscribing to the same $i3$ ID, as shown in Figure 5. Since each middlebox sees more or less the same packets (the set of packets might be slightly different due to packet loss and delay), each middlebox will end up maintaining roughly the same state for the connection.

To make this solution transparent to the receiver, we need to filter out replicas before delivering them to the receiver. This can be achieved by having both the primary and the backup middleboxes send their packets to another middlebox whose only function is to filter out duplicate packets. One problem with this proposal is that the filtering middlebox can fail as well. Fortunately, we can easily deal with this failure by using the anycast functionality as shown in Figure 4. The key observation here is that no connection state needs to be transferred from the primary to the backup filtering middlebox, and thus no other recovery technique is needed.

The main advantages of this solution are that it provides fast recovery from failures, and it is transparent to end-hosts. The obvious disadvantage is the overhead, as each packet is replicated and processed separately.

Ping-Pong Storage. Another solution that allows fast recovery is storing the connection state in the packets exchanged by the end-hosts. For instance, the middlebox can create a special control packet that is sent back and forth between the end-hosts (hence the name ping-pong storage). Whenever this packet passes through the middlebox, the middlebox stores the current connection state in the packet. In case of failure, the backup middlebox simply restores the connection state upon receiving this packet.

While this solution requires changes to end-hosts, these changes are minimal as end-hosts do not need to interpret the state in the packet. If the connection state is small enough, the state can be piggybacked in the data and acknowledgment packets, if any. The downside of this method is that it is not appropriate for cases where the middlebox stores a considerable amount of per connection state (*e.g.*, the last I frame in an MPEG stream).

5. CONCLUSIONS

In this paper, we argue that $i3$ is a natural platform for service composition. The key feature of $i3$ that enables service composition is the ability of end-hosts to forward packets through a given set of middleboxes. In addition, the $i3$ anycast and multicast primitives enable applications to implement service discovery, middlebox selection, and transparent rerouting in case of middlebox failure.

To support legacy applications and middleboxes, we have developed a proxy that transparently tunnels packets over $i3$. In ongoing work at UC Berkeley, we leverage the $i3$ and proxy functionality to provide intrusion detection as a service. At the University at Tübingen, $i3$ service composition is used to bridge heterogeneous networks and devices, and provide security to mobile users.

However, there are two important issues that $i3$ does not address directly: recovery from middlebox failure when middleboxes maintain connection specific state, and path selection. We discuss these issues in detail, and suggest several approaches to address them. Fully addressing these issues represent an interesting and challenging subject of future work.

6. REFERENCES

[1] P. Chandra, Y.-H. Chu, A. Fisher, J. Gao, C. Kosak, T. E. Ng, P. Steenkiste, E. Takahashi, and H. Zhang. Darwin: Customizable Resource Management for Value-Added Network Services. *IEEE Network*, 15(1), 2001.

[2] Y. Chen, D. Bindel, H. Song, and R. H. Katz. An Algebraic Approach to Practical and Scalable Overlay Network Monitoring. In *Proc. SIGCOMM*, 2004.

[3] X. Gu, K. Nahrstedt, R. N. Chang, and C. Ward. QoS-Assured Service Composition in Managed Service Overlay Networks. In *Proc. of ICDCS*, May 2003.

[4] L. Guo and I. Matta. Search Space Reduction in QoS Routing. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 41(1):73–88, 2003.

[5] B. Karp, S. Ratnasamy, S. Rhea, and S. Shenker. Spurring Adoption of DHTs with OpenHash, a Public DHT Service. In *Proc. of IPTPS*, 2004.

[6] K. Lakshminarayanan, A. Rao, I. Stoica, and S. Shenker. Flexible and Robust Large Scale Multicast using *i3*. Technical Report CS-02-1187, UCB, 2002.

[7] A. Nandi. LALA : Location Aware Load Aware Overlay Anycast. MS Thesis, Rice University, May 2004.

[8] T. S. E. Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *Proc. of INFOCOM*, 2002.

[9] V. Paxson. Bro: A System for Detecting Network Intruders in Real-time. *Computer Networks*, 31(23–24):2435–2463, 1999.

[10] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *SIGCOMM*, 2002.

[11] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. of ACM SIGCOMM*, 2001.

[12] D. Xu and K. Nahrstedt. Finding Service Paths in Multimedia Service Proxy Network. In *MMCN*, 2002.