Core: A Peer-To-Peer Based Connectionless Onion Router

Olaf Landsiedel, Alexis Pimenidis, Klaus Wehrle
Department for Computer Science
RWTH Aachen, Germany
firstname.lastname@cs.rwth-aachen.de

Heiko Niedermayer, Georg Carle
Computer Networks and Internet
University of Tuebingen, Germany
firstname.lastname@uni-tuebingen.de

Abstract

Onion Routing is today's typical substrate for anonymous near-real-time communication. Via layered encryption, Onion Routers such as TOR [7] or Tarzan [8] build a static tunnel through a peer-to-peer relay network. All traffic exchanged between two end points uses one and the same tunnel, making the design susceptible to attacks based on pattern analysis. Recent publications [12] even extend this theoretical threat by showing the practical feasibility of a pattern analysis attack on the deployed TOR system.

In contrast to today's anonymous communication systems, Core routes each packet a different communication path and so is not susceptible to this class of attacks. Furthermore, inspired by IP-routing the connectionless approach reduces the complexity of the Onion Router.

In this paper, we describe the design of our Connectionless Onion Router, evaluate its performance, and address the communication overhead. We present address virtualization to abstract from Internet addresses and to provide transparent application support. Thus, no application-level gateways or proxies are required to sanitize protocols from network level information. Acting as an IP-datagram service, our scheme provides a substrate for anonymous communication to a wide range of applications using TCP and UDP.

1 Introduction

Increasing censorship and the erosion of privacy in digital communication and particularly the Internet challenges the freedom of speech. Individual communication and private information is becoming subject to monitoring and surveillance of government and corporate institutions. Around the world, governments have tried and often succeeded in blocking access to web sites and information considered subversive and not suitable. Even the actions of European and US governments raise questions about privacy and civil liberties; for example, the Federal Bureau of Investigations surveillance system Carnivore, the Patriot Act, and the European Union's "Convention on Cybercrime". These give the authorities in the US and the European Union a wide range of powers to intercept, log, and record personal digital communications.

The research community proposed Onion Routing [17] as an answer to such challenges to everybody's privacy. Using Onion Routing, a host can connect to a server through a set of mix relays and thus hide its identity. Layered encryption ensures that each hop in the relay network can only decrypt the address of its successor in the relay chain.

When a node communicates to a server, it uses a static route through the mix network. As this route remains the same until the communication between the server and the node has ended, the connection-based design of today's Onion Routers enables attacks against the user's anonymity [12] or the recipient's anonymity [13]. To

attack a connection, the adversary applies a traffic pattern to one of the relays of the connection. This pattern then interferes with the communication between the server and the node and can thus be measured at both of them.

Instead of using a static route between two nodes, Core routes each packet along a different path to its destination – making pattern attacks, as the one described above, useless. Furthermore, the connectionless design is inspired by the IP-routing in the Internet. Thus, a relay does not maintain state and flow information for the connections as in TOR. This reduces the complexity of the design and the architecture, allowing for small memory footprints and the ease of implementation of Core nodes.

The remainder of this paper is structured as follows. First, section 2 addresses related work and discusses the differences to our approach. Section 3 gives a deeper insight into TOR and the pattern-based attack. Next, section 4 presents Connectionless Onion Routing as our solution to this class of attacks and discusses the design challenges we faced. Section 5 discusses anonymous services and section 6 addresses threats to anonymity. Next, section 7 evaluates the performance of the proposed approach. Section 8 discusses future work and concludes the paper.

2 Related Work

In this section, we address the shortcomings of today's near real-time anonymous communication schemes. Their design principle is the Chaumian Mix [4], in which e-mail traffic is forwarded through a set of cascading mixes to hide the sender's identity. The traffic enters one of the mixes and leaves the mix network at some random point. To reduce the impact of malicious mixes, the route through the mixes is set up via layered encryption. Thus, each mix only can decrypt the information about its successor in the cascade. Based on this design, various systems have been proposed enabling near real-time communication for services like web browsing: Onion Routing [17], TOR [7], Freedom [3] and Web Mixes [2]. They use a static set of relays and therefore suffer from drawbacks like limited scalability and fault tolerance. Furthermore, they are easy to attack via Denial of Service or legal approaches, as the set of mixes is well known. Systems like MorphMix [14] and Tarzan [8] avoid this problem since they are based on peer-to-peer networks. However, they suffer from the dynamics of an overlay network and the limited bandwidth each node can provide. All the above mentioned systems rely on static routes and are thereby susceptible to pattern-based attacks.

To our best knowledge, our work is the first to propose Connectionless Onion Routing, i.e. to route each packet along a different path for low latency anonymous networks. The closest work to Core is a theoretical paper by Serjantov and Murdoch [15] about splittinglarge messages in remailer systems [6, 11]. It analyzes the usage of independent routes for each packet of a message to prevent an eavesdropper at the first mix from determining the message size and therefore being able to correlate this with an eavesdropper located near other mix nodes. As nodes are repeatedly used as first relay in a path, the packet flow is exponentially distributed to prevent the first relay from estimating the message size. This theoretical analysis primarily refers to remailer systems. However, in private communication, its authors confirmed that most arguments also apply to onion routing.

3 Background

In this section, we present a deeper insight into "The Onion Router" (TOR) and the pattern-based attack. It is necessary to discuss both in more detail as the attack against anonymity is one of the motivations for our work.

3.1 Understanding TOR

The TOR network is designed to provide anonymous TCP connections to arbitrary Internet servers. The network consists of hundreds of nodes that relay traffic and thereby hide the stream's origin. Data flow in TOR is similar to conventional circuit switched networks, as each TOR node represents a switch. The initiator of a TCP stream

creates a TOR circuit by connecting to a first randomly selected TOR node (relay). With a Diffie-Hellmann key exchange, both negotiate a secret key and thereby establish a secure channel. Via the relay, the initiator can connect to further TOR nodes and thus build a chain of relays. Commonly, TOR uses three relay nodes.

Messages are encrypted multiple times (layered encryption). Each relay removes one layer of encryption to determine its successor and to change the cipher text. Thus, no correlation of incoming and outgoing streams is possible. Furthermore,the layered encryption ensures that each node only knows its successor on the path.

The last TOR node in the chain is the exit point. Via this node, the initiator can connect to standard web servers or set up ssh sessions to arbitrary nodes in the Internet.

3.2 Attacking TOR

After discussing TOR's architecture, we introduce the pattern-based attack [12] in this section as an example of an practical and feasible attack on this network. All connections relayed by a TOR node share its limited bandwidth and processing power. For example, when one node relays traffic for two other nodes, they share the available bandwidth equally. Thus, relayed connections interfere with each other. To attack the TOR network, the adversary needs to control a TOR node (the threat model allows for malicious TOR nodes) and send a specific traffic pattern to a selected TOR relay. As the bandwidth of the relay is shared, this traffic pattern will interfere with the existing streams of the node. Therefore, the pattern can be retrieved with conventional traffic-analysis techniques [5] from the nodes that relay the attacked streams. Furthermore, for the same reason it can be seen at the source and destination of the stream. Thus, communication in TOR is not anonymous anymore.

The attack is based on the fact that TOR nodes have limited bandwidth which is shared between all relays. Therefore, any participating node can perform this attack, making it possible even for adversaries with few resources to threaten anonymous communication in the TOR network.

4 Introducing Core

This section introduces our solution to the attacks presented in the previous section. In particular, we discuss the architecture of the Connectionless Onion Router.

Today's Onion Routers operate connection oriented. Thus, when the tunnel is set up, each relay stores session information, e.g. symmetric keys, state information, successor, and predecessor. As Core operates connectionless, such a session setup cannot be done[1] and is unnecessary[2]. Therefore, in Core, each packet needs to maintain the information about all relays in path and the corresponding asymmetric keys.

Core uses elliptic curve cryptography based Diffie-Hellman key exchange to generate symmetric keys for decrypting the information about the successor on each relay. Assume that the sender has generated a public / private key pair and that it knows the public key of the relay. From its own private key and the relay's public key, it can retrieve a shared secret, i.e. a symmetric key. With this key, it encrypts the information about the next hop(s) and the destination. By adding its own public key to the header, it ensures that the relay itself can retrieve the symmetric key and determine its successor on the path (see Fig. 1). By applying this scheme to each hop, Core ensures that each relay knows only about its successor. Additionally, padding packets to constant length prevents relays from identifying the length of the path and their position in it.

To reduce the computational and the bandwidth overhead we use elliptic curve cryptography (ECC). Compared to RSA, ECC keys are much shorter. We use a key length of 192 bit, which has a security level similar to 2048 bit RSA keys. Additionally, encryption, decryption, key generation, and elliptic curve based Diffie-Hellmann key exchange (ECDH) are significantly faster than in RSA-based schemes [10].

[1] This is good for security reasons.

[2] This is good to achieve a small memory footprint.

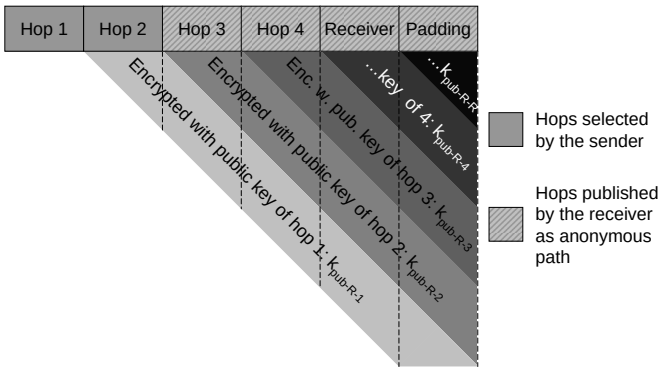

Figure 1: Path information: similar to onion routing, the layered public key encryption ensures that each hop can only determine its successor.

Additionally, the sender adds a return route – leading back to itself – to the packet. Using this return route, the receiver can send an answer back to the sender. When forwarding a packet, a relay stores the previous and next relays and the corresponding key for some minutes to prevent the reuse of routes. This is important as the repeated use of a route would enable the pattern attack and so threaten the anonymity of the path destination.

4.1 Path Concatenation

So far, we discussed how a sender can communicate with a known receiver. However, we did not discuss how the receiver itself can stay anonymous.

We propose a path concatenation scheme to enable receiver anonymity. An anonymous service (hidden service) publishes a route to itself in some out-of-band media (see section 4.2). The sender retrieves this information and adds its own hops to the route by adding more layers of encryption to the path section. For example, assume that Bob offers an anonymous service, that Alice wants to communicate to this service, and that she retrieved a route to the anonymous service. First, Alice selects a number of random nodes to relay her traffic in order to stay anonymous herself (see Fig. 2). Next, Alice concatenates the nodes she selected and the communication path she retrieved. Thus, she selected the head and Bob the tail of the path (see Fig. 1) and both stay anonymous.

Furthermore, the Core design has to ensure that nobody can impersonate an anonymous service and offer – potentially malicious – own services instead. We use public / private key-based signatures to protect against impersonation. Thus, Bob computes a key pair and the public key becomes his anonymous identity. When Bob publishes a path section, he signs it with his private key. Knowing Bob's anonymous identity, a Core node can verify the issuer of the path sections.

4.2 Service Directory

As discussed in the previous section, Core needs a lookup service to provide paths to anonymous services. In Core, the service directory provides this information. To hide its identity, a Core node communicates with the service directory anonymously. In more detail, it provides the following services:

- **Relay discovery:** each Core node can retrieve other Core nodes from the directory and use these to relay traffic. Furthermore, it registers itself as relay. Retrieving Core nodes can be done anonymously or directly to enable bootstrapping.
- **Virtual IP addresses:** to address anonymous services, each Core node can retrieve a virtual IP address (e.g. 10.2.3.79) from the service directory. This IP address is unique in the Core system and other nodes can use it to address and to connect to a service.

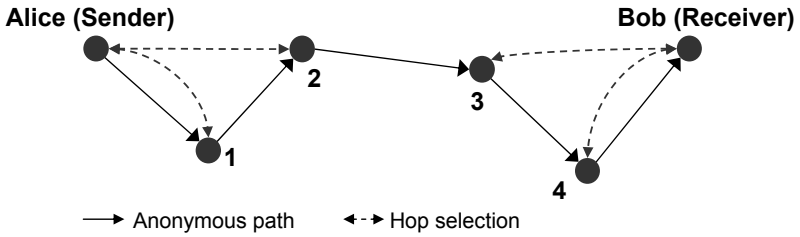

Figure 2: Sender and receiver independently select a number of hops on the anonymous communication path.

- **Name Service:** next to virtual IP-addresses, we allow Core nodes to register a domain name.
- **Path sections:** to connect to a service, the sender needs to retrieve a path section pointing to the desired service. Thus, a service can store such path sections in the service directory.

The service directory can either be a highly scalable and redundant peer-to-peer network (e.g. a DHT) or a set of dedicated machines. When communicating with the service directory, a Core node signs its messages to prove its anonymous identity.

4.3 Key Caching

Core benefits from elliptic curve cryptography as its keys are shorter and cryptographic operations are significantly faster than traditional RSA. These two factors make the design of a Connectionless Onion Router possible. However, ECC is still slower by an order of magnitude than symmetric key operations [10]. Thus, Core caches keys to reduce the number of ECC operations drastically.

Packet forwarding consists of two operations. First, a relay computes the symmetric key via Diffie-Hellmann key exchange and then it decrypts the packet with this symmetric key. Of these two operations, the Diffie-Hellmann key exchange is the more computationally expensive one, as it bases on asymmetric cryptography. Thus, Core tries to reduce the overhead of this operation.

When a Core node uses a relay again, at any point in the relay path, it will reuse the same key set with a certain probability. Thus, the sender and the relay cache the asymmetric and symmetric keys for a while. When the key is reused, the sender and the relay do not need to recompute the symmetric key. Our evaluation (see section 7) shows that key caching can drastically increase the throughput of the Core system. Note that this technique allows the relays, sender and receiver each two distinctive choices: sender and receiver have the choice to decide themselves between performance and confidentiality, while relays have the possibility to choose a tradeoff between processing power or memory. While both choices are independent, we expect this feature to have a very positive impact on the deployment.

4.4 Legacy Support

For a fast adoption of Core by a wide user range, it is necessary that the user can reuse his unmodified networking applications in the Core environment. Commonly, networking applications use TCP and UDP protocols, which depend on the IP protocol. To provide support for such legacy applications Core resides on the network level and is transparent to applications, i.e. Core provides an IP overlay to the application.

It is necessary that the outgoing packets of a Core node do not contain any information about its real identity, e.g. its IP address. Other anonymous communication schemes use application-level gateways or proxies, but this requires the gateway or proxy to know the protocol formats of all used applications. Due to the large number of exiting application level protocols we do not consider this engineering effort reasonable. Furthermore, some

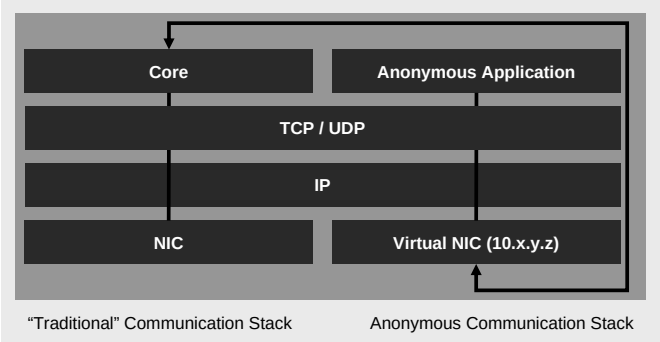

Figure 3: The virtual network interface enables transparent application support.

protocols are proprietary and their formats are not published. We address this problem by using an “anonymous communication stack” which is assigned an virtual IP address, e.g. `10.2.3.79` (see Fig. 3). The essence of this address virtualization technique is to abstract away any information about the identity of a node while remaining compatible with the IP addressing scheme and applications that depend on it.

We use the service directory (see section 4.2) to assign unique virtual IP addresses to each node. As an alternative, we discussed deriving the node’s IP addresses from its public key. However, the IP address space is too small to avoid collisions which renders this approach infeasible.

4.5 Naming Service

For a seamless integration of existing applications, a name service is required. Users need a way to address anonymous services and to distinguish them from Internet-based ones.

Thus, we provide a name service, which is transparent to the existing Domain Name Service (DNS), in order to map between DNS requests and anonymous services. Thus, users can enter `http://www.freespeech.anon` into their web-browsers and connect to an anonymous service. When a `.anon` domain is queried, the system sends a DNS request in order to resolve this domain. Our scheme reroutes this request to the Core service directory and determines the virtual IP address and corresponding communication paths from the service directory. To distinguish between traditional domain names and the anonymous ones, we introduce new domain names, like `.anon`, as used in the previous example.

5 Core and Internet Services

In this section, we discuss how services can be deployed in the Core network. We distinguish two classes of services: (1) anonymous services and (2) exit points.

As Core is an IP overlay, a Core node can provide the same services as a server in the Internet. Thus, it can run a web server, ftp server, ssh, content distribution, or streaming services. Introducing the virtual `.anon` domain makes it very intuitive for the user to connect to an anonymous service.

Furthermore, a node can act as exit point. Exit points bridge between the Core network and the Internet itself. Therefore, a Core node can communicate anonymously to any Internet based service, for example a web-site, via an exit point. An exit point can provide access at two different protocol layers: (1) it can operate at the network level and act as NAT device or (2) it can operate at application level, for example as an HTTP-proxy. As Core operates is accessed from the applications via the virtual NIC, standard off-the-shelf http-proxies and NAT software can be used. Additionally, we allow the operator of an exit point to introduce their own exit policies by restricting protocol types, destinations, and consumed bandwidth.

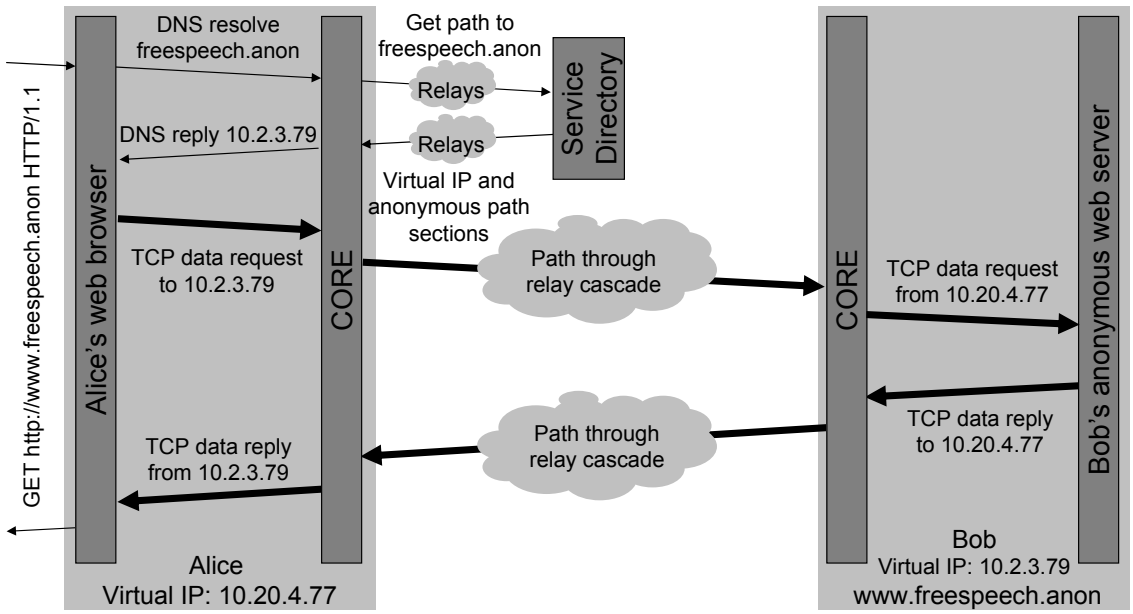

Figure 4: Seamless integration of existing applications into the anonymous communication scheme: in this example, Alice uses a web browser to request the site `http://www.freespeech.anon` from an anonymous web server.

Compared to many other existing anonymous communication schemes [8, 14], Core provides a seamless integration of anonymous services and exit points with the overall architecture.

5.1 Communication Example

Connecting to a service in the Core architecture consists of two steps: (1) retrieving a path to the service and (2) communicating to the service itself (see fig. 4). In this section, we discuss these two steps in more detail. Let Alice connect to the service `www.freespeech.anon` and assume that Alice knows some Core nodes that can relay traffic for her.

Alice enters `www.freespeech.anon` into her web browser. As a result, the web browser sends a DNS query to resolve the domain name. In the network stack, the Core application intercepts all queries for the `.anon` domain and thereby ensures that the query does not leave the host and leaks information about Alice's intentions. Instead of communicating to a standard DNS server, the Core application queries – via a number of relay hosts – the service directory. From the directory, it retrieves the virtual IP address of `www.freespeech.anon` and some paths to it. After retrieving this information, the Core application sends a fake DNS reply back to Alice's web-browser. Next, the web-browser initiates a number of TCP connections to the web-site. As the destination address is a virtual IP address, packets are routed via the anonymous communication stack into the virtual NIC. Thus, the Core application relays them anonymously into the CORE network.

6 Analyzing Core's Security

In this section, we analyze the security of the proposed anonymous communication scheme. We present our threat model and discuss threats to anonymity and privacy.

6.1 Threat Model

Commonly, a global passive adversary is the assumed threat when analyzing theoretical anonymity designs. However, like all practical anonymous communication schemes that are scalable and enable low-latency communication, Core can not protect against such a strong adversary. Furthermore, similar to its predecessors, Core does not prevent (usually long-term) intersection attacks by local adversaries.

We assume a practical adversary which can:

- observe a limited part of the network
- participate in the Core network with a limited number of relays.
- participate actively by offering a service, e.g. running a web server.
- influence communications by generating, delaying, dropping, and modifying traffic content and patterns.

6.2 Passive Attacks

In this class of attacks, the attacker does not actively participate in the Core network.

Content observation: as communication traffic is encrypted, its observation does not reveal content.

Packet size correlation: packet size correlation is not possible in Core, as all packets are padded to equal length.

Source / destination observation: as traffic is relayed and packets do not include any information about the number of hops, the first hop can not decide solely from receiving a packet if its predecessor is the source or a relay of the packet. Similarly, the last relay can not determine whether it is the last hop.

Packet counting: in our usage scenario data rates and the amount of data are individual for a particular communication. Therefore, an observer could try to correlate this information to follow a flow through the network. However, as Core does not build tunnels through the network but sends each packet on a different path, intermediate nodes do not recognize the packets of one flow as such and, thus, can not count the packets of one flow.

When compared to the security provided by Tor, we see that Tor fails to provide security, if the first and last relay of a circuit work together to identify the originator of a data stream. This is not only true for accessing external ressources through the network, like e.g. accessing the WWW, but also for access to hidden services [13]. None of these attacks can be applied to Core, as the path is selected randomly for each packet, the attacker only sees a fraction of the traffic. Furthermore, it is recommended that Core nodes do not uniformly select first or last relays, but unevenly according to some statistical distribution. Serjantov and Murdoch propose to use an exponential distribution [15] in their work on anonymous remailers. As a consequence, the relays can not estimate the real amount of data that is sent or received. Such an approach is important when the amount relayed traffic is small, e.g. new nodes are only known by few nodes.

Core is also not susceptible to classical predecessor attacks [18] due to its onion routing, which distinguishes it from Crowds. Note that there are currently no research results on the impact of onion routing on this class of attacks.

Pattern observation: although traffic observation does neither reveal content nor sender and receiver, it may reveal patterns. The arguments here are similar to the arguments for the packet counting attack. First of all, we can state that flows are not sent over a constant path because each packet follows an individual, randomly selected path. Patterns are therefore heavily distorted by the different delays of the various paths. Compared to existing solutions, Core drastically reduces the possibility of application and data-specific patterns being revealed.

6.3 Active attacks

In this class of attacks, the adversary participates actively in the Core network.

Controlling relays: controlling multiple relays is probably the most dangerous attack. The multi-hop routing results in a high probability of a packet traversing at least one compromised hop. An attacker which owns all relays for a packet knows sender and receiver. However, a relay does not have any knowledge about its position in the relay path nor the path's length. Thus, the attacker must use statistical methods to determine the sender-receiver relationships. Such an attack is only possible with either many compromised relays or with long-term, high-volume communication relationships. To further mitigate this attack, nodes can select hops with diverse IP addresses for a path. The assumption is that it is difficult for an adversary to compromise nodes with addresses in different IP ranges [8].

Furthermore, nodes should not use long-lived identities, which is unnecessary for regular users of the system, who can easily use session-based identities. An exception are anonymous services, which have to be reached via their anonymous ID. An attacker could use multiple relays and active communication with a service to attack it. Methods a service could use to mitigate the attack are using location-diverse relays, having a set of (more) trusted relays, and non-static IP addresses for the service.

Offer a service: an adversary may offer an anonymous service. The service itself can not directly determine the user as she protects her identity with her own path. However, this attack could be used by an attacker to attract a particular set of senders to itself and its paths. This does not directly threaten the anonymity of the users, but could be used in combination with other attacks.

Pattern insertion: an attacker may insert a pattern into the network in order to find the pattern in other places of the network. As stated in the section about the passive pattern observation, such an attack is difficult in practice as flows are only visible at the true end points of a communication and thus patterns can only be flow specific if added directly at the end points. Furthermore, sending each message on a completely different path and each node relaying high amounts of traffic makes it harder to detect patterns than in other existing systems.

Marking, tagging and replacing: end-to-end encryption and integrity checks prevent attacks on marking, tagging, and replacing messages.

Replay attacks: replay attacks are addressed by frequent key changes as well as short time message hashing. Nodes store a hash of each relayed message over a certain amount of time. Incoming messages are hashed and only relayed when the hash has not been seen before.

6.4 Attacks against the service directory

Most of the attacks against the service directory are denial of service attacks. An attacker could continuously request paths to a service and therefore make it inaccessible as paths can only be used once.

Other attacks include the impersonation of an anonymous ID. However, Core mitigates this attack by signing the communication with the service directory with an anonymous identity (see section 4.2).

6.5 Attacks by the service directory

When a nodes registers itself as relay at the service directory, it needs to reveal its IP-address – otherwise it cannot be used as relay. However, all other communication – including the assignment of virtual IP-addresses – with the service directory is done anonymously through the Core network (see section 4.2). Thus, the service directory cannot map a virtual IP-address to the actual IP-address of the corresponding node.

6.6 Forward Secrecy

Core does not support forward secrecy. Forward secrecy means that breaking a long-term secret, e.g. a private key, does not allow to decipher recorded messages as the derived shared keys cannot be computed. The Diffie-Helmann key exchange is the basis for adding forward secrecy to asymmetric cryptography. In Core however,

the Diffie-Hellmann secrets are solely used as public / private key pairs. Thus, Core does not provide forward secrecy. Nonetheless, frequent key changes mitigate the danger of compromised keys. Additionally, asymmetric cryptography can be expected to be secure for several years.

6.7 Concluding the security analysis

Core is designed to prevent pattern analysis and packet counting attacks inside the network. It improves the anonymity in comparison to other systems for interactive near-real-time traffic in these crucial points.

Important measures to achieve this anonymity in addition to the basic design idea are:

- path length unlimited and random
- uneven distribution of flows among first or last hop
- location/IP-diverse paths
- avoid long-lived identities

7 Evaluating the Core Architecture

In this section, we evaluate the proposed architecture. First, we introduce implementation details, and then present throughput measurements. Finally we discuss our ongoing work of deploying Core in the "wild".

Core, in contrast to existing anonymous near real-time communication schemes, relays each packet via a different path (Connectionless Onion Routing), which results in interesting implementation challenges:

- **Bandwidth and performance overhead:** when each packet is routed along a different path, no relay tunnel can be set up and no connection establishment is done. Therefore, no shared secrets can be bound to a tunnel. Instead, Core has to rely heavily on asymmetric cryptography, which is computationally expensive. Furthermore, as each hop through the mix cascade requires a key to be embedded in a packet's header, Core needs to deal with additional bandwidth overhead.
- **Round-trip time variance:** routing each packet along a different path results in a highly dynamic communication scheme. Some packets will be on routes with low round trip times due to proximity or low load while other packets will have high round trip times. This results in packets being delivered out of order at the final receiver. Since TCP misinterprets this out-of-order delivery as packet loss, its fast retransmission mechanism leads to additional packets and a decreased overall throughput.

7.1 Implementation

We implemented Core on Linux. A Core node consists of two parts: (1) the relay service and (2) the virtual network interface. For elliptic curve cryptography, we use the Sizzle code [9] which recently became part of the OpenSSL library.

To relay a packet in the Core network, it needs to contain information about the next relay on the path, e.g. its IP address and the port number. Furthermore, the packet needs to contain the key the decrypt all this information, i.e. to remove one layer of encryption.

IP address and port account for six bytes. With the 192 bit ECC key and one reserved byte, this information fits into a 32 byte header. Thus, Core's overhead is 32 bytes per relay, which we consider reasonable. Additionally, Core uses a cell size of 1000 bytes to further reduce the header overhead.

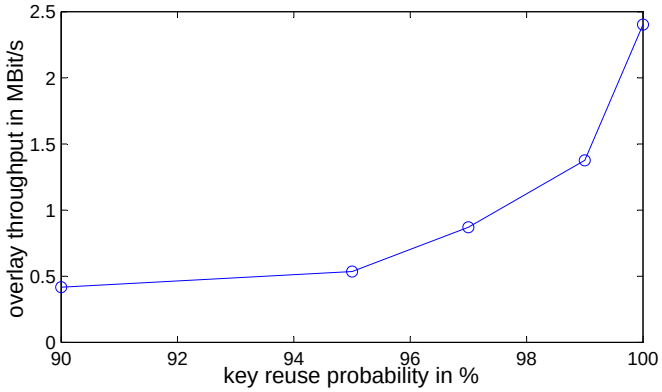

Figure 5: Key caching allows a throughput of up to 2.5 MBit/s.

7.2 Performance

We deployed Core on our test network which consists of eight Linux hosts (5x Pentium IV 2.2 GHz, 3x Xeon 2.8 GHz) running Fedora Core 4, interconnected by a 100 MBit/s network.

Each host acts as relay and one of the hosts offers an anonymous service. Another one communicates to this service via TCP. For the evaluation, sender and receiver use three relays each, resulting in a total of eight hops. The overlay throughput, i.e. the net throughput that an application running on top of the Core overlay can use, is depicted in figure 5. The evaluation shows that key caching can drastically speed up the throughput of Core as it reduces the need for asymmetric cryptography. When making full use of key caching, the throughput reaches 2.5 MBit/s. Furthermore, our measurements show, that the throughput is limited by the two factors: (1) a large number of TCP retransmission which are caused by the high round trip time varience and (2) CPU processing performance, where the cryptographic operations mainly cause the processing load.

7.3 Core in the Wild

Currently, we are deploying Core in the wild on PlanetLab [1] hosts all over the world. Running a TCP stream over the Core overlay on PlanetLab is challenging. Due to the high CPU load and bandwidth utilization of the PlanetLab nodes [16], many packets are dropped. The ones that successfully make their way through the overlay have round trip times up to several seconds, while others are routed on a fast path and reach their destination in less than a second. As a result, TCP interprets this out-of-order delivery as packet loss and causes fast retransmissions, which further decrease the performance.

We evaluated that (1) packet play-out smoothing to limit out-of-order delivery and (2) per-hop acknowledgements reduce packet loss and thereby increase TCP performance. Nonetheless, it can be said that TCP was not designed to cope efficiently with such dynamic overlays and round trip times. Currently, we are focusing on the design and implementation of a flow control algorithm that can cope with such highly dynamic systems and provide high troughputs.

8 Future Work and Conclusion

In this paper, we present a novel approach to anonymous Internet communication: Connectionless Onion Routing. Core routes each packet along a different path in an IP overlay. Compared to existing Onion Routers, Core is not susceptible to pattern-based attacks. Its transparent architecture and name service allows a seamless integration of existing applications without changes to the applications or communication protocols. Core is implemented

transparently in the network stack via virtual IP addresses.

Elliptic curve cryptography and key caching reduce the header and performance overhead drastically. Our evaluation shows that Core can reach a throughput of up to 2.5 MBit/s. However, our PlanetLab experiments have shown that routing TCP streams over the highly dynamic Core overlay is challenging and proposes some open research questions on which our current work focuses.Future work includes amongst others support for some forward secrecy by adding a second public / private key pair for encryption only.

References

[1] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating System Support for Planetary-Scale Network Services. In *Proc. of Symposium on Networked Systems Design and Implementation (NSDI)*, March 2004.

[2] O. Berthold, H. Federrath, and S. Köpsell. Web MIXes: A system for anonymous and unobservable Internet access. In *Proc. of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, July 2000.

[3] P. Boucher, A. Shostack, and I. Goldberg. Freedom Systems 2.0 Architecture. White paper, Zero Knowledge Systems, Inc., Dec. 2000.

[4] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of ACM*, Feb. 1981.

[5] G. Danezis. The Traffic Analysis of Continuous-Time Mixes. In *Proc. of Privacy Enhancing Technologies workshop (PET)*, May 2004.

[6] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proc. of the IEEE Symposium on Security and Privacy*, May 2003.

[7] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proc. of 13th USENIX Security Symposium*, August 2004.

[8] M. J. Freedman and R. Morris. Tarzan: A Peer-to-Peer Anonymizing Network Layer. In *Proc. of 9th ACM Conference on Computer and Communications Security (CCS)*, Nov. 2002.

[9] V. Gupta, M. Millard, S. Fung, Y. Zhu, N. Gura, H. Eberle, and S. C. Shantz. Sizzle: A Standards-Based End-to-End Security Architecture for the Embedded Internet. In *Proc. of IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2004.

[10] V. Gupta, D. Stebila, S. Fung, S. C. Shantz, N. Gura, and H. Eberle. Speeding up Secure Web Transactions Using Elliptic Curve Cryptography. In *Proc. of the Network and Distributed System Security Symposium (NDSS)*, 2004.

[11] U. Möller, L. Cottrell, P. Palfrader, and L. Sassaman. Mixmaster Protocol — Version 2. Draft, July 2003.

[12] S. J. Murdoch and G. Danezis. Low-Cost Traffic Analysis of TOR. In *Proc. of the IEEE Symposium on Security and Privacy*, May 2005.

[13] L. Overlier and P. Syverson. Locating Hidden Servers. In *Proc. IEEE Symposium on Security and Privacy (SP)*, May 2006.

[14] M. Rennhard and B. Plattner. Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection. In *Proc. of Workshop on Privacy in the Electronic Society (WPES)*, Nov. 2002.

[15] A. Serjantov and S. J. Murdoch. Message Splitting Against the Partial Adversary. In *Proc. of Privacy Enhancing Technologies workshop (PET)*, May 2005.

[16] N. Spring, L. Peterson, A. Bavier, and V. S. Pai. Using PlanetLab for Network Research: Myths, Realities, and Best Practices. In *Proc. of Second Workshop on Real, Large Distributed Systems (WORLDS)*, December 2005.

[17] P. F. Syverson, D. M. Goldschlag, and M. G. Reed. Anonymous Connections and Onion Routing. In *IEEE Symposium on Security and Privacy*, Feb. 1997.

[18] M. K. Wright, M. Adler, B. N. Levine, and C. Shields. The predecessor attack: An analysis of a threat to anonymous communications systems. In *ACM Transactions on Information Systems Security*, 2004.