CADET: Confidential Ad-hoc Decentralized End-to-End Transport

Bartlomiej Polot
Technische Universität München
Munich, Germany
Email: polot@net.in.tum.de

Christian Grothoff
Technische Universität München
Munich, Germany
Email: grothoff@net.in.tum.de

***Abstract*—This paper describes CADET, a new transport protocol for confidential and authenticated data transfer in decentralized networks. This transport protocol is designed to operate in restricted-route scenarios such as friend-to-friend or ad-hoc wireless networks.**

We have implemented CADET and evaluated its performance in various network scenarios, compared it to the well-known TCP/IP stack and tested its response to rapidly changing network topologies. While our current implementation is still significantly slower in high-speed low-latency networks, for typical Internet-usage our system provides much better connectivity and security with comparable performance to TCP/IP.

I. Introduction

Existing IP networks are not suitable for secure, decentralized ad-hoc networking applications. IP routing requires trusted routers to assign structured addresses to their clients, and at higher levels BGP requires business contracts to negotiate peering relationships. Ad-hoc community networks require protocols that avoid the resulting overheads in planning and business negotiation, and eliminate the insecurities resulting from the dependency of network users on network operators.

This paper presents *CADET*, a new algorithm for establishing robust end-to-end transport-layer connections in completely self-organized networks without a central authority. Starting with an arbitrary network topology (such as a wireless mesh, a physical LAN or the peering relationships between autonomous systems), the algorithm uses modern "friend-to-friend" distributed hash tables (DHTs) to discover a redundant set of available paths, creates multiple, switched *connections* between the endpoints and then uses those to create a robust *tunnel* for end-to-end encrypted, authenticated communication. Multiple applications can then multiplex TCP or UDP-like *channels* over the tunnel.

A fundamental design principle of CADET is key-based routing (KBR). Addressing systems by their public key eliminates the need to use a network protocol to obtain a network address, and thus eliminates the use of insecure protocols like RARP or DHCP, or manual management processes by which users are assigned addresses out-of-band. A system which uses a public key as its address has the fundamental benefit of being able to cryptographically prove its ownership of the address, as only it has knowledge of the respective private key. However, using public keys as addresses creates the problem of routing messages to the respective address, as public keys have no address space structure that would be suitable for making routing decisions.

Many existing overlay networks use distributed hash tables (DHTs) to locate values by hash. While traditional DHTs assumed an underlying (IP-based) routing layer, more recent designs operate over an arbitrary mesh topology. CADET uses a randomized friend-to-friend DHT to discover multiple paths to the target system.

CADET can be viewed as a peer-to-peer (P2P) implementation of the TCP/IP protocol: it allows a node in a decentralized and unstructured P2P network to find and connect to any other peer in the network, and to confidentially exchange authenticated data. Like SCTP, CADET supports transmitting multiple reliable and unreliable channels over a single connection.

II. Background

A. 802.11s

In ad-hoc wireless networks each node typically creates its own routing tree. One of the most well-known implementations of mesh networks is IEEE 802.11s, which is designed to be used in small area wireless networks. The standard does not require the use of a particular routing protocol — any suitable protocol may be used. However, the hybrid wireless mesh protocol (HWMP) must be supported by any node that complies with the standard. HWMP is based on the ad-hoc on-demand distance vector protocol (AODV) [16] and tree-based routing. The key property of the protocol is that it simultaneously builds paths proactively, and on-demand. This provides good performance for known nodes (as the paths can be pre-built), and accommodates node churn by offering an additional on-demand mechanism for nodes that recently joined the network and are thus not yet well known.

Another routing algorithm that can be used in 802.11s networks is B.A.T.M.A.N. [13] In B.A.T.M.A.N., each node broadcasts its existence; this information is re-broadcasted by its neighbors with an increased hop count. Each node remembers the hop count and neighbor the message came from, putting the neighbor with the smallest hop count in the routing table. This way each node is aware of each other node in the network and has a first hop information to reach it. Naturally, this approach does not scale to larger networks and is problematic if there is significant churn. The re-broadcasting can also lead to traffic amplification which might be used for

denial of service attacks. Existing B.A.T.M.A.N. deployments are limited in size and are thus not affected by the scalability problems.

B. F2F DHT

Distributed hash tables (DHT) are a useful tool to perform routing in restricted topologies, such as mesh networks or friend-to-friend networks. In a typical DHT, information is identified by a key and stored at the peer in the network whose ID is closest to the key. To do that, each DHT has a routing algorithm that directs each request, identified by a key, to the corresponding peer. This is sometimes called key-based routing (KBR). While most popular DHTs (Kademlia [10], Pastry [19]) require a well-connected group of peers to perform this routing correctly, there are some designs that work in the absence of unrestricted connectivity.

One such DHT is X-Vine [11]. Just like in Chord [21], X-Vine maintains a so-called finger table. In X-Vine, a finger is a sequence of peers needed to reach a destination. When discovering peers close in the ID-space from existing peers in the neighbor table, not only the ID of the new peer is shared, but also the finger information. This is joined with the finger towards the peer who shared the information, forming the complete finger towards the new peer.

A particular problem in DHTs is the Sybil attack [6]. In this attack, an adversary creates a great number of identities and joins the network in order to disrupt it. DHTs follow different approaches to mitigate this attack: Persea [1] divides the ID space among a set of bootstrap nodes and assigns each one management right over its *chunk*. Each peer in the network can invite other peers, assigning them an ID as well as delegating control over a subset of its own chunk. This way even if an attacker obtains an invitation to the network via social engineering, it could not choose it's IDs freely and would be limited to the chunk assigned by the inviting peer.

A different Sybil-resistant DHT, Whanau [9] has proposed using the social graph and random walks to counter this threat, although at the cost of high control overhead.

Another DHT that can deal with Sybil attacks and restricted routes is R^5N [7]. $R5N$ uses a recursive variant of Kademlia for its routing table and mitigates attacks by randomizing the destination peers and paths used to get there. Specifically, random neighbours are chosen for the first $O(\log n)$ routing steps where n is the size of the network to insure widespread replication. Afterwards, instead of soliciting routing information to determine the peer closest to the key, R^5N greedily forwards requests to the closest neighbors. When a peer is closer to the target than any of its neighbors, R^5N assumes this to be the destination and performs the requested DHT operation (i.e. store or lookup). The operation is thus carried at the peer *locally closest* to the key, not necessarily the one *globally closest*. To compensate for this, requests are replicated to target several locally closest peers. The combination of randomization, replication and local choices makes R^5N robust against active adversaries.

III. Design

CADET is expected to operate above a restricted communication layer with properties similar to those of Ethernet or WLAN, and in conjunction with a DHT that can operate in the resulting restricted-route network.

Given these two lower-layer components, the roles of both IP and TCP/UDP are then fulfilled by CADET. In particular, CADET takes care of the connectivity needs of peers that cannot establish direct connections. CADET uses the DHT to discover routes, tries to optimize them and performs authentication, encryption and traffic control between any two peers in the network. This should always succeed as long as the peers are in a connected subgraph of honest participants.

CADET itself is organized in three layers. The bottom layer provides connectivity, like IP. The middle layer provides end-to-end authenticated encryption, similar to TLS. The top layer provides multiplexing, traffic control and other optional features, such as reliability, in a way similar to TCP, UDP or SCTP.

A. Connectivity

The bottom layer of CADET provides connectivity between two endpoints. By endpoints we understand peers that are running the applications that are going to communicate with each other. All the other peers may function as *relays*: peers that participate in a communication but are neither the origin nor the destination of the traffic. Relays simply forward messages from one neighbor to another.

The connectivity layer has two important concepts: *paths* and *connections*. A path is simply a sequence of peers, where each pair of peers from the sequence is directly connected (on the lower, Ethernet-like layer below CADET). A path is thus just information about the topology of the network stored at an individual peer; it does not carry any communication with other peers. It is roughly the equivalent of a phone number in the telephone system. On the other hand, a connection is a *reserved* path that is in use; creating a connection from a path is thus equivalent to the process of using a phone number to create a phone circuit when a call is made. When an origin peer wants to communicate with another peer in the network to which it cannot connect to directly, the origin peer discovers a path between the two and notifies all the peers in the path about its intention to communicate with the destination, thereby creating a connection.

Each connection has a unique 256 bit random ID. Each peer on the path needs to store the connection ID, together with information about the next and previous peer it should relay traffic to. Two peers can have more than one connection between each other; in fact, this is common to improve reliability and performance. Whenever multiple connections are available, they are used simultaneously by sending messages on the connection that has the least load at the time.

1) Path discovery: In order to discover paths to other peers, CADET uses two sources of information:

- Passive monitoring of connections being established by other peers. Here, CADET simply analyzes the traffic

it relays for other peers and incorporates the topology information contained in connection requests into its own local view of the network.
- Explicit DHT requests, where the DHT is instructed to record the route taken by GET and PUT requests. More specifically, each peer periodically makes a PUT request to the DHT with its ID and information that may help to establish a direct connection to it (such as lower layer addressing information). When CADET tries to connect to a peer to which it does not know any path, CADET issues a GET request using the ID of the destination peer. By joining the PUT route and the GET route, CADET obtains a path towards the destination peer. CADET naturally also tries to make use of the information from the DHT to try to establish a direct connection.

2) Connection establishment: When a peer decides to establish a new connection a destination peer, it first checks if there are any known paths towards the destination. If there are no paths, the service initiates a DHT query to find them.

When there is at least one known path, the origin peer creates a connection with a random 256 bit ID and sends a $CONNECTION\_CREATE$ message containing a list of all the peers in the path to the first peer on the path. Each peer forwards the message to the next peer, while storing the connection details to be able to forward subsequent messages identified just by the connection ID. At the same time, each peer uses the path to passively learn potential paths to reach the origin, the destination, as well as all the intermediate peers in the connection.

The destination receives the incoming connection and sends a $CONNECTION\_ACK$ message to the origin peer in order to confirm the correctness of the path. Each peer on the path uses the stored information from the creation message to send the message in the opposite direction. At the same time, all the potential paths to peers participating in the connection are considered confirmed.

The first peer sends another $CONNECTION\_ACK$ to confirm the receipt of the first $CONNECTION\_ACK$ and finish the connection creation. When a peer is expecting a $CONNECTION\_ACK$, it starts a timer to retransmit the message it sent (either $CONNECTION\_CREATE$ or $CONNECTION\_ACK$) in case the message or the acknowledgement was lost.

3) Keepalive: In order to avoid saturating the network with connections, idle connections are destroyed after a timeout. There are two idle counters, one for each direction, as any endpoint being down is a reason to tear down the connection. To avoid this timeout, peers periodically send keepalive traffic on idle connections.

4) Congestion Control: To avoid saturating intermediate peers with traffic, peers use an ACK mechanism for congestion control on each hop of a connection. Each peer has a dedicated buffer per connection and sends an ACK to the previous peer in the connection when the buffer has room for new messages. Since the lower layer is not expected to guarantee reliable delivery, CADET uses a polling system to compensate for lost data or ACK messages.

B. Security

The second CADET layer provides authentication and encryption for the communication, encapsulating all traffic in *tunnels*. A tunnel is a secured communication session between two peers. The tunnel uses connections to send data to remote peers, with a redundancy goal of having three independent connections at any time. This redundancy serves both reliability and performance purposes. The first messages exchanged by two peers are a key exchange in which they authenticate each other and use emphemeral keys establish a session key. This session key is cycled periodically and once it is changes it becomes impossible to decrypt captured traffic, even if the endpoints are seized and forensically analyzed. Additionally, each message uses a random initialization vector to prevent the same transmitted plaintext from rendering the same cyphertext and thus leaking information to an eavesdropper.

1) Tunnel establishment: When two peers first connect (and periodically after that), they perform a handshake to establish the tunnel's encryption keys. Every peer in the network maintains a Curve25519 [2] ephemeral key EdDSA-signed [3] with the peer's permanent identity key. The ephemeral key is changed periodically every 12 hours, thus every tunnel's keys change on average every 6 hours. Once the ephemeral key changes, all captured traffic that used the old ephemeral keys is no longer decryptable, therefore providing forward secrecy [4].

To establish a tunnel, each peer sends the other endpoint an initial ephemeral key message containing its identity (the peer's public key), the ephemeral key, the ephemeral key's validity period and a signature to validate the data. Both peers send this in parallel, without waiting for each other.

On receipt of an ephemeral key message, each peer checks the signature and validity period. If everything is correct, CADET derives the key material from both ephemeral keys using ECDH [2]. This key material is fed to a key derivation function [8], together with the peers identities and a salt value to obtain the symmetric keys used to encrypt payload traffic in the tunnel. The symmetric keys are 512 bits which are split into a 256 bit AES key and a 256 bit Twofish key.[1] CADET uses different symmetric keys for sending and receiving; these are obtained by feeding the identities of the peers in different order to the key derivation function.

As soon as the symmetric keys are obtained, each peer sends a challenge PING message to the other peer, encrypted with the new keys containing the remote peer's identity and a nonce.

To verify the correctness of the key exchange each peer checks its own identity in the PING message and, if it is correct, sends the PING's nonce in a corresponding PONG response message. Upon receipt of the PONG message, CADET checks the nonce before it considers the tunnel established.

[1] We expect AES to be virtually free, as it is often implemented in hardware. However, we want to avoid trusting the hardware implementation and thus also encrypt with Twofish. As the two symmetric keys are independent, the resulting security should be at least as good as the stronger of the two cipher implementations.

C. Multiplexing

CADET uses fast Curve25519 public key cryptography [2], but it is still important to minimize the use of this operation. To avoid triggering a handshake every time two applications on a pair of peers communicate, CADET multiplexes all communication channels for a given pair of endpoints inside one tunnel. A *channel* is a communication stream between two applications running on peers participating in the network. They can be on the same or different peers. In case they are on different peers, the messages sent to each other are transmitted in a CADET tunnel to the other peers, which demultiplexes them and delivers them to the corresponding application. In order to share a tunnel among multiple channels, each channel has a unique ID. CADET offers optional, per channel options, such as out of order delivery and reliability.

1) Channel establishment: Channel establishment happens in a similar way to the connection establishment. First, the application requests a new channel, with the destination specified by a peer ID and a port number. It identifies the channel with a sequential 32 bit local ID with the most significant bit set to 1. The local ID is unique per client and many clients on a computer may use the same local ID.

CADET checks if there already is a tunnel towards the destination peer. If there is no tunnel, it creates one as described previously.

Given an existing tunnel, the origin peer creates a channel with a sequential 32 bit ID and sends a `CHANNEL_CREATE` message on the tunnel. The ID has the most significant bit set to 0 and the second most significant bit depends of the relation between the public keys of the endpoints to ensure the IDs generated by the endpoints do not collide with each other.

The destination receives the incoming channel and checks the port number for local applications listening on that port. If an application is listening, it sends a `CHANNEL_ACK` message to the origin peer in order to confirm the channel and notifies the application about an incoming channel. Otherwise it sends a `CHANNEL_NACK` to reject the channel creation.

The first peer sends another `CHANNEL_ACK` to confirm the receipt of the first `CHANNEL_ACK` and finish the connection creation.

If the first peer receives a `CHANNEL_NACK`, it destroys the channel and notifies the application about the failure.

When CADET expects a `CHANNEL_ACK`, it starts a timer to retransmit the message it sent (either `CHANNEL_CREATE` or `CHANNEL_ACK`) to handle the case where the message or the ACK was lost.

2) Flow Control and Reliability: Similar to TCP streams, CADET offers reliable channels. CADET uses message ACKs for flow control on these channels, with a fixed window size of 64 messages. If a message is lost but subsequent messages are received, this can be indicated, since each ACK message has a 64 bit mask to signal "future" received messages. This allows the sender to retransmit only the messages that are really lost while sending subsequent messages in the window. Messages confirmed as received are not stored anymore and the timing is used to adjust the retransmission delay for subsequent data.

Currently the delay is calculated as the average of the last eight round-trip times, although different methods are possible and this choice may be revisited in the future.

Unreliable channels, in a similar fashion to UDP, have no flow control, although they benefit from the congestion control, since it is done on the connection level.

3) Communication session: Now that we have presented all CADET components, we describe how a typical CADET communication session is established. An example session with all elements can be seen in Figure 1

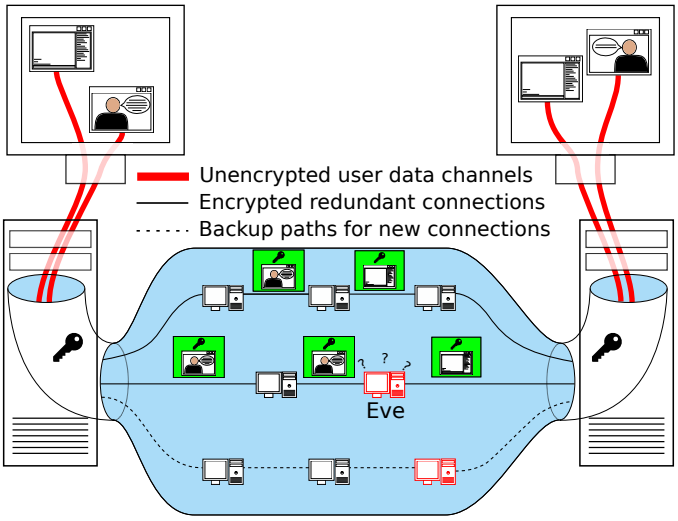

Fig. 1: Two channels inside a tunnel sending messages over two connections with one known backup path.

First a client application requests a new channel to a certain destination peer and port number. The destination peer is defined by its public key and the port number is a 32 bit number.

CADET receives the request and checks whether there is an existing connection. If there is already an established connection towards the destination peer, this connection is used. Otherwise a new connection is established.

Then CADET checks for an existing tunnel towards the destination peer. If there is no tunnel, CADET creates the it. Both endpoints use the existing connections to perform a handshake. In the handshake the peers authenticate each other and establish the session keys to communicate securely. If there is a tunnel when the request arrives (for instance, because another application running on the peer is already communicating with the remote peer, or did so recently), the existing tunnel is used.

The service creates a new channel inside the tunnel, allocating a unique number and specifyng the destination port. If there is an application on the destination peer listening to the given port, the channel is created.

The two proccesses can communicate on the new channel by using the newly created channel. The tunnel is shared between

all channels and each message sent uses one of the available connections.

If either of the endpoints does not need the channel anymore, it requests its destruction. The service notifies the other endpoint about the channel's destruction.

When the last channel inside a tunnel is destroyed the service starts a timer for the tunnel. After the timeout, the service destroys the tunnel, deleting the encryption keys and destroying all established connections.

IV. Implementation

We have implemented CADET in the GNUnet P2P framework.[2] Similar to TCP/IP, GNUnet is divided in several layers, roughly equivalent to those in TCP/IP, as seen in Figure 2. At the bottom we have the *Transport* layer, which provides the most basic communication: moving bits from A to B. Transport supports various plugins (TCP, UDP, HTTPS, WLAN, Bluetooth, etc.) and selects a good one automatically.

On top of Transport, the *Core* layer is the GNUnet layer directly comparable to Ethernet in the TCP/IP model. Core creates direct "logical" links between two peers and (unlike Ethernet) provides authentication and encryption. In GNUnet, a direct link is any link that Transport can create. If Transport cannot establish a link (for instance, due to firewall policy, NAT gateways or range limitations for radio transmissions), Core will also be unable to establish a link.

GNUnet currently uses the R^5N DHT for routing GET and PUT requests in the resulting restricted environment. Together, these three layers thus satisfy the design requirements for CADET. On top of CADET, we are building other GNUnet subsystems, like a VPN, file sharing or Conversation (a voice application).

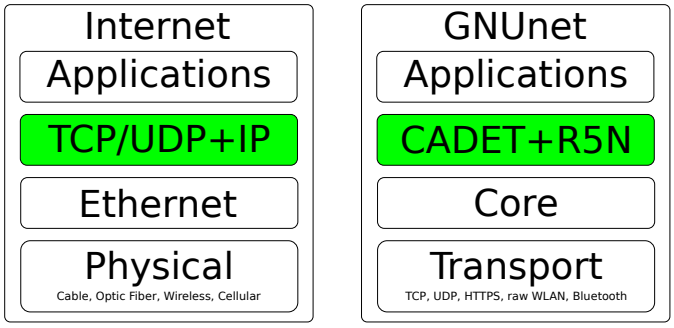

Fig. 2: On the left we present the current TCP/IP stack. On the right we show roughly equivalent components in the GNUnet system.

V. Results

We evaluated CADET using the implemenation in GNUnet and GNUnet's testbed infrastructure [22] which can be used to deploy, control and observe thousands of peers on a workstation PC. For the tests, a small network with 100 peers was used. Each of the experiments was repeated 10 times with a different random underlay topology being emulated each time. We introduce an artificial 20 ms round-trip delay on Transport level links, to simulate a high-speed network.

All experiments were done running the normal GNUnet production code, including operating system scheduling, cryptographic operations and network I/O.

A. Churn Resistance

In the connectivity test we start a network with all peers online and an average of 21 random connections per peer. We select 10% of the peers to ping another randomly selected peer. Then we start churning the network decreasing the number of active peers to 80% of the original number, followed by further decreases by 10% every 10 s until we reach 20%. We just churn non-participating (relay) peers, and never disrupt peers doing or receiving pings. It is possible that at the end, when only peers sending and receiving pings are left, some of them might be disconnected from the network.

We contemplate three different scenarios. In the most favorable one, peers are allowed to freely connect to other peers as they see fit. This would correspond to Internet peers with public IP addresses and no blocking firewall. In a more restricted scenario we do not allow peers to establish new connections, but before starting the test we "warm up" the peers: we establish a random connection from each peer on the network to another random peer. This allows peers to observe some `CONNECTION_CREATE` messages for which they are relays, and thereby learn a bit about the topology of the network. In the most unfavorable scenario, we start the network and right away start the benchmark, thus including all operations that help discover and optimize routes in the observed cost. We choose to only perform pings with 10% of the peers so peers do not learn too much of the topology from each other's connections; otherwise the differences between the "warm" and "cold" cases might be harder to observe.

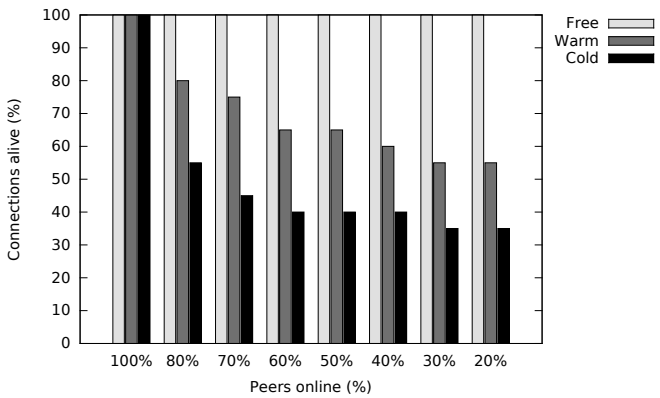

Fig. 3: Percentage of peers that keep their connections as the network is churned. The test labelled as "Free" peers are allowed to establish new connections. In the "Warm" test the peers have time to learn the topology of the network prior to the churn. In the "Cold" test, peers have to find a route to their destination as the network is started.

[2]https://gnunet.org/

In Figure 3 we see that in all three scenarios every peer initially manages to find a route to their destination. As we churn the network peers start to disconnect; however, the results vary. In the case of "Free" peers, they never lose connectivity because they are permitted and thus quickly manage to establish direct Transport connections to their targets. As they no longer need relays, they remain unaffected by churn. In the restricted route scenarios we observe that knowledge about the topology accumulated during the warm-up phase allow peers in the "Warm" scenario to keep their connectivity much longer, due to knowledge of alternative paths and the redundant connections that it allows. Even with 80% of the network gone, more than 50% of peers manage to stay connected to their targets. In the worst case "Cold" scenario, almost 40% of peers still manage to stay connected after losing 80% of the network in just a minute.

	Free	Warm	Cold
Avg	44.26 ms	49.18 ms	53.39 ms
std dev	13.97 ms	17.93 ms	18.64 ms
first round	55.99 ms	55.87 ms	62.38 ms
last round	38.52 ms	35.44 ms	44.98 ms

TABLE I: Average and standard deviation of latency among peers running under different network conditions (disconnected peers are excluded from the average).

Additionally, Table I shows the average latency between the peers. Here, it is important to remember that the test was run with 10 ms latency on each link between peers. As expected, "Warm" peers that have gathered more information about the network topology are able to achieve connections with shorter paths earlier, which reduces latency. In the case of the "Free" scenario, the lack of initial information is compensated by establishing new, direct connections. Looking at the latency in the last round, we can observe why the peers without the extra initial information lost connectivity more frequently: their average connection is significantly longer, therefore more vulnerable to being disrupted by disconnected relay peers.

B. Latency

In order to measure the impact of the additional architectural layers on latency, we measured the round trip times to machines running the normal unmodified GNUnet code on different types of networks, as depicted in Figure 4. First we tested three types of networks with computers with direct connectivity: in a gigabit LAN with a direct cable connection (Peers $A-C$), in a 300 Mb/s wireless LAN on the same access point (Peers $A-B$) and between a university network and a typical home 16 Mb/s (down) and 1 Mb/s (up) ADSL Internet connection (Peers $A-D$). The fourth scenario involves two computers directly connected with a cable ($A-C$), of which only A has Internet connectivity. The computer with no Internet connectivity (C) pings the peer only reachable over the Internet (D), using A as a relay. The control results are obtained with a simple ICMP ping. Each connection was evaluated using 150 round-trip measurements. The results are presented in Table II.

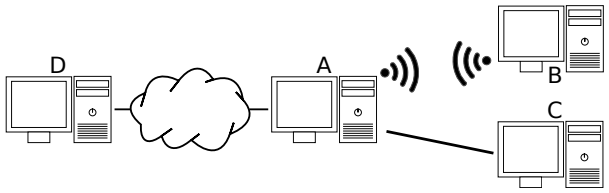

Fig. 4: The test setup used for performance measurements. Peers B and C have no Internet connectivity.

	ICMP		CADET	
	Avg	Std Dev	Avg	Std Dev
LAN	0.21 ms	0.04 ms	14.31 ms	4.24 ms
WLAN	5.02 ms	3.85 ms	26.23 ms	15.05 ms
WAN	37.94 ms	1.01 ms	44.66 ms	1.32 ms
Relay	N/A	N/A	56.38 ms	5.02 ms

TABLE II: Average latency between systems running GNUnet on different connections. The latency penalty, relative to the absolute latency, is less significant over Internet connections.

The LAN connection has the biggest proportional latency penalty, since the ICMP case is very close to hardware performance and nearly instantaneous. Traffic in GNUnet has to travese multiple layers in userspace and has the overhead of encryption. This causes the latency to be 14 ms compared to the 0.2 ms of ICMP. The absolute penalty is even bigger in a wireless environment. The larger packet size of the GNUnet ping probes causes collisions in the access point and the forced retrasmissions increase the latency 21 ms over the ICMP case, and the standard deviation spikes to 15 ms. When the test is performed over a home Internet connection the proportional impact is much smaller. In this case the pings over GNUnet are only 6.5 ms longer with similar variance.

The reason why CADET increases latency by 14 ms in the LAN experiment (compared to the ICMP baseline) but only 7 ms in the WAN case can be explained by the computing power available: while the local area computers B and C are laptops the computer D is a workstation. Cryptographic operations (and in particular ECHE) are significantly slower on B and C, which explains the difference in this case.

Finally, when we ping the workstation from a computer without Internet connectivity, we obtain 56 ms, which is a latency 3 ms shorter than the LAN + Internet latencies combined. This is due to the fact that the relay node does not need to perform (re)encryption on the traffic on the CADET level, although it does on Core level.

C. Throughput

In Table III, we present the results of our throughput tests. Using the setup from the latency measurements, we transferred

several files between the computers. For the LAN test the files were 100 MB. For the wireless LAN we used 100 MB and 10 MB files. For test over the Internet — due to the slow 1 Mb/s upload speed of the ADSL line — we used 1 MB files. Each transfer was repeated 10 times. Between iterations, the GNUnet tunnels were not torn down and thus only the first transfer is affected by the full handshake (connection and tunnel establishment); all subsequent transfers benefit from tunnel reuse. However, the channels inside the tunnel were re-established for every transfer.

To compare pure TCP speed we used the program `netcat`. For CADET we used `gnunet-cadet`, a custom `netcat`-equivalent command line tool to access GNUnet's CADET implementation.

	TCP	CADET
LAN	105 MB/s	15.0 MB/s
WLAN	4.97 MB/s	5.01 MB/s
WAN	114 KB/s	103 KB/s
Relay	N/A	110 KB/s

TABLE III: Average throughput in the different network types.

For the high speed LAN test the throughput for GNUnet was much smaller than with raw TCP. This is caused by the high CPU utilization for cryptographic operations and user space overheads. During these tests the CPU was operating at 100% while for raw TCP the load was not noticeably higher than idle. A faster CPU would help the GNUnet stack offer better performance results for LAN speeds. On the wireless LAN tests both protocols offer similar results, as is the case on the Internet tests: when the CPU is not saturated we see that CADET performs about as well as TCP.

Surprisingly the test for Relay traffic yields slightly better throughput compared to the WAN connection. We repeated this test, with similar results. While not fully explained, the difference is small, so it is conceivably due to fluctuations on the ADSL connection outside of our control.

In these experiments CADET's flow control did not seem to negatively impact performance. However, additional experiments (including traffic loss and message reordering) may be done in the future to further refine the design.

VI. Related Work

A. TCP/IP

TCP/IP [17] shares most of the functionality with CADET. The three distinct functions (connectivity, state maintenance and multiplexing) are divided in just two layers, with IP providing connectivity, TCP or UDP multiplexing and the state maintenance shared between the two, with each connection defined by a 5-tuple of IP addresses, TCP/UDP ports and protocol. This causes any change in the connectivity (switch to a different address, for instance) to break all established communications. In CADET all application level communication happens in channels inside a tunnel, which is independent from the connection used to send the data, therefore connections can and do change without affecting the application.

Regarding the individual components, IP solves the connectivity problem using hierarchical routing. This requires an authority to assign the addresses and a coordinated protocol to spread the routing information over the network, which GNUnet in general and CADET in particular, avoid as a design principle. IP addressing does not only create security problems, but also implies the need for routing algorithms (such as BGP or OSPF) to work; as the IPv4 address space is exhausted, fragmentation causes the routing tables to grow exponentially in size. While IPv6 addresses some of this issues, it fails to address key issues such as ownership, man-in-the-middle attacks or mass surveillance.

TCP solves the reliability problem with byte-oriented ACKs while CADET uses per-message ACKs, since GNUnet's Transport layer does not deliver fragmented messages to the Core layer. The biggest difference, however, is the flow and congestion control. TCP uses a *sliding window* for flow control and different *congestion windows* for congestion control. The basic mechanism is additive increase and multiplicative decrease in the window size. The window grows slowly until a duplicated ACK is received, which means a packet was lost, most probably due to a router dropping it due to a full buffer. The reaction to this depends on the implementation, but usually implies reducing the window size by a big step and resuming the slow increase. Since UDP does not use ACKs, it lacks flow or congestion control; this again can then be abused. Since CADET uses ACKs for congestion control, it is guaranteed that it will never drop a message due to full buffers. This also allows the unreliable, UDP-like mode to benefit from congestion control and a rudimentary flow control. The extra traffic generated by the hop-by-hop congestion control ACK is minimized by using bigger message sizes (64 kB vs typically 1500 bytes for IP/Ethernet), which minimize the overhead used by the ACKs.

A related transport protocol, SCTP [20], designed as part of the TCP/IP stack, implements some improvements like message-oriented transmissions, multihoming, multistreaming and individual selection of features like in-order delivery and reliability. However, it still does not offer payload security and still suffers from the problems associated with IP routing.

The Host Identity Protocol [12] is a proposed additional layer in the TCP/IP stack to separate the host addresses from the host identity. Using host identity tags (public key hashes) to identify a host, it allows the IP address to change while maintaining any TCP connections established. The protocol adds a Diffie-Hellman exchange to each TCP connection and allows for authentication and optional payload encryption.

B. Tor

While Tor [5] also uses P2P relay nodes to forward data between endpoints, the goals of the systems are totally different. Tor uses forwarding to achieve annonymity and requires all nodes to be able to communicate with one another to create random paths among all the peers in the network. CADET

operates in environments with limited connectivity and uses forwarding to improve connectivity, therefore often using the shortest path it can find. Tor uses onion encryption to hide the full paths from each relay peer, while CADET keeps them in the clear such that other peers can cheaply learn about the topology of the network and optimize their own connections.

C. net2o

net2o [15] also uses source routing, but instead of establishing a connection by explicitly allocating resources at every relay peer on the path, net2o uses a bitmap in the message header to tell every hop which neighbor to send the message to. Each hop consults this bitmap and retransmits the message to the neighbor whose number is indicated in the field corresponding to it. This way the relay does not need to store any infromation regarding the connections that traverse it. On the other hand, the source peer must know all the neighbors of all the relays on the path, as well as find the path itself, which is not trivial and the solution proposed by net2o has exponential complexity. While this method might be possible in an infrastructure setting where the neighbors of a core router rarely change, the dynamic nature of P2P scenarios make this impracticable, as it would require constant updates to the peering information of each relay.

D. Level 2 systems: PSTN, ATM, MPLS

CADET connections are similar to circuits in the traditional telephone system, virtual circuits in ATM [14] or label switched paths in MPLS [18]. In all of these designs, all relay nodes must be informed of a connection before they forward payload traffic. A significant difference between CADET and the traditional layer 2 designs is how routing is performed: PSTN uses hierarchical phone numbers to route calls. ATM and MPLS use external protocols based on shortest-path-first to perform the routing (PNNI- and OSPF-based protocols, respectively). ATM uses fixed 48 bytes of payload with 5 bytes of routing information. This causes the need to use additional mechanisms to transport payload, like the AAL protocols. The small message size causes the routing overhead to be as much as 10%. CADET allows messages of up to 64 kB, which can help lower the relative overhead of the headers. In MPLS and CADET data can travel on different paths. Unlike MPLS, the CADET connection identifier/label is global and does not change at every hop.

VII. Conclusion

We have presented CADET, a secure decentralized transport for ad-hoc networks. CADET provides a secure communication channel by combining active DHT searches with passive monitoring to achieve good connectivity in most scenarios.

We have shown that — while having notably higher latency and resource utilization compared to TCP in fast, high-bandwidth networks — in Internet-like scenarios normal end-users are likely to experience today, the performance differences are small. We also have demonstrated that CADET adapts quickly to dramatic changes in the topology of the network, as it may be necessary for P2P or ad-hoc networks.

Acknowledgments

This work was funded by the Deutsche Forschungsgemeinschaft (DFG) under ENP GR 3688/1-1.

References

[1] M. N. Al-Ameen and M. Wright, "Persea: a sybil-resistant social dht," in *Proceedings of the third ACM conference on Data and application security and privacy*. ACM, 2013, pp. 169–172.

[2] D. J. Bernstein, "Curve25519: new diffie-hellman speed records," in *PKC*, 2006.

[3] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," *Journal of Cryptographic Engineering*, vol. 2, no. 2, pp. 77–89, 2012.

[4] N. Borisov, I. Goldberg, and E. Brewer, "Off-the-record communication, or, why not to use pgp," in *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, ser. WPES '04. New York, NY, USA: ACM, 2004, pp. 77–84.

[5] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," DTIC Document, Tech. Rep., 2004.

[6] J. R. Douceur, "The sybil attack," in *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*. London, UK: Springer-Verlag, 2002, pp. 251–260.

[7] N. Evans and C. Grothoff, "R5n: Randomized recursive routing for restricted-route networks," in *5th International Conference on Network and System Security*. Milan, Italy: IEEE, 2011, pp. 316–321.

[8] B. Kaliski, "PKCS #5: Password-Based Cryptography Specification Version 2.0," RFC 2898 (Informational), Internet Engineering Task Force, Sep. 2000.

[9] C. Lesniewski-Lass and M. F. Kaashoek, "Whanau: A sybil-proof distributed hash table," in *7th USENIX Symposium on Network Design and Implementation*, 2010, pp. 3–17.

[10] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*. Springer-Verlag, 2002, pp. 53–65. [Online]. Available: http://portal.acm.org/citation.cfm?id=646334.687801

[11] P. Mittal, M. Caesar, and N. Borisov, "X-vine: Secure and pseudonymous routing using social networks," *arXiv preprint arXiv:1109.0971*, 2011.

[12] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson, "Host Identity Protocol," RFC 5201 (Experimental), Internet Engineering Task Force, Apr. 2008, updated by RFC 6253.

[13] A. Neumann, C. Aichele, and M. Lindner, "B.a.t.m.a.n status report," The Better Approach To Mobile Ad-Hoc Networking Project, Tech. Rep., June 2007, http://open-mesh.net/batman.

[14] T. S. S. of ITU., *ITU-T Recommendation I.150: Integrated Services Digital Network (ISDN) General Structure : B-ISDN Asynchronous Transfer Mode Functional Characteristics*. International Telecommunication Union, 1993.

[15] B. Paysan, 2009. [Online]. Available: http://net2o.de/internet-2.0.html

[16] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," RFC 3561 (Experimental), Internet Engineering Task Force, Jul. 2003.

[17] J. Postel, "Transmission Control Protocol," RFC 793 (Standard), Internet Engineering Task Force, Sep. 1981, updated by RFCs 1122, 3168, 6093, 6528.

[18] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture," RFC 3031 (Proposed Standard), Internet Engineering Task Force, January 2001.

[19] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware 2001*. Springer, 2001, pp. 329–350.

[20] R. Stewart, "Stream Control Transmission Protocol," RFC 4960 (Proposed Standard), Internet Engineering Task Force, Sep. 2007, updated by RFCs 6096, 6335.

[21] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4. ACM, 2001, pp. 149–160.

[22] S. H. Totakura, "Large scale distributed evaluation of peer-to-peer protocols," Masters, Technische Universitaet Muenchen, Garching bei Muenchen, 06/2013 2013.