TARZAN: Techniques of a peer-to-peer anonymizing network layer with security analysis from a more recent perspective

- Advanced seminar: Peer-to-Peer Networks -

March 16, 2007

Steffen Schott
Faculty of Applied Sciences, Freiburg, Germany
schott@informatik.uni-freiburg.de

supervised by:

Arne Vater and Prof. Dr. Christian Schindelhauer,
Chair of Computer Networks and Telematics

CoNe
Freiburg

Contents

1 Abstract

In November 2002 Michael J. Freedman and Robert Morris presented their paper *Tarzan: A peer-to-peer anonymizing network layer* on the ACM Conference of CCS 2002. From that (particular) time onwards, various other papers have examined the security of Tarzans proposed anonymizing system and functionality. This essay summarizes the techniques applied in Tarzan to achieve anonymity and gives an analysis of security aspects concerning these techniques. As I was not able to find any further novelties about Tarzan or improvements presented by the authors, all assumptions presented in this essay are based on the description of the network in [FM02a].

2 Introduction

The main goal, that Freedman and Morris were trying to achieve, is the anonymity of a sender or receiver contacting some other source on the Internet. For example if a user wishes to query a website without disclosing his identity.

To achieve this, the user wants neither to depend on an honest webserver nor the webserver to participate in any kind of anonymizing network. Nobody should be able to discover the user's identity. This means, Tarzan primarily protects one of side of a communication channel, although both sides can be protected if they are using Tarzan. As well, the given approach is supposed to be applicable for more than just HTTP. Given that Tarzan is a network layer protocol substituting the ordinary IP-layer, it becomes clear, that basically every higher-level protocol can be tunnelled and therefore anonymized by Tarzan. As shown in figure1, ordinary data packets do not go higher than the network layer in the OSI abstraction layers when being relayed. As a result, any higher-level protocol, including TCP and UDP, are tunnelled in the same manner by Tarzan. To satisfy the expectation of not being specifically vulnerable to a global adversary, this approach is fully peer-to-peer. Such an adversary might be capable of taking over or blocking any kind of centralized structure, for example core routers.[1]

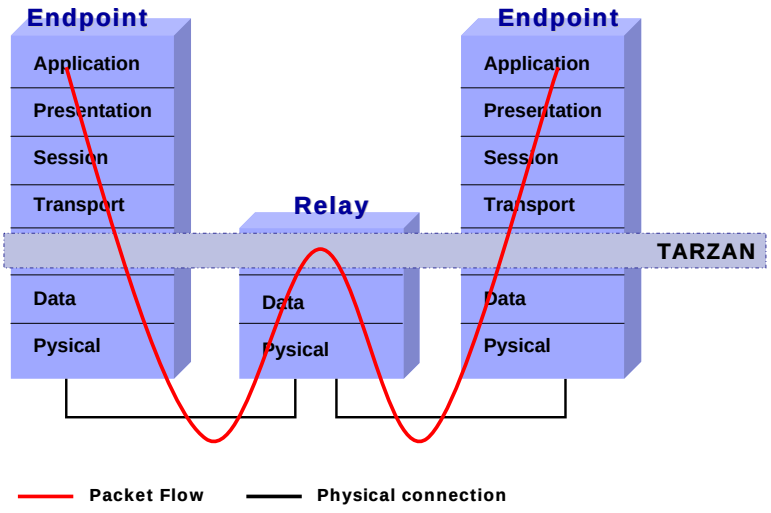

Figure 1: OSI abstraction layers with packet flow and Tarzan layer.

For a Tarzan participant to build up a secure tunnel the following architecture is applied. Not only is Tarzan fully peer-to-peer to avoid liability of a central instance. The initiator at setup of a tunnel successively chooses peers for the purpose of forward his traffic at random from a set of nodes. This set is determined for each step on the basis of a hash table. As hashing is configured to change on a day to day basis and peer selection is partly at random, the choice of peers for the

[1] Some interesting examples of possible vulnerabilities of for example Centralized Mixnets are shown in [FM02b].

tunnel is hard to foresee. Still, for example the openness respectively the own IP address can cause serious problems for anonymity, as we will see in subsection 4.2. Furthermore cover traffic and Onion Routing style encryption are used. Cover traffic on the one hand, minimizes possibilities of traffic analysis attacks, while, on the other, Onion Routing style encryption helps to avoid content disclosure and traceability of path.

When a user wants to send a data packet to a particular webserver on the Internet, he sends the packet through the tunnel. From the endpoint of the tunnel, the initiator's query then will be forwarded to the final destination and therefore leave the Tarzan network. As this endpoint of the tunnel translates all packets Source-IP addresses to its own IP address the endpoint is called a *pseudonymous network address translator* (PNAT). As a result of forwarding the query, the identity of the PNAT is exposed to the webserver and any type of adversary.

In the following, I will describe the employed techniques in detail and point out possible as well as prevented threats in section 3. In section 4, I then wish to pick up these threats and discuss possible and prevented attacks, before discussing possible improvements and concluding and in section 5 and 6.

3 Applied Techniques

As indicated in section 2, various techniques are applied to achieve anonymity for the system as a whole. Since every technique is responsible for protecting the network against a specific type of adversary, I want to point out threats and means of protection in the following subsections.

3.1 Sanitising Headers

During the encrypted tunnelling of data packets no relaying peer can retrieve further details from higher layers of the OSI abstraction layers, as shown in figure1. However at the end of the tunnel data packets are decrypted and all information up to the application layer disclosed. Therefore if the last node of the tunnel is an adversary, it can read all header information and the payload, should the latter not have been additionally encrypted. The same applies to the destination webserver or any global eavesdropper, that could evaluate and collect data from the packet's payload. To hinder this kind of information disclosure, higher-level headers are sanitised by Tarzan and the Tarzan layer substitutes the assigned IP-address with a randomly chosen one, just like a NAT. The node at the end of the tunnel realises the same kind of 'NATing' from the randomised IP address to his private IP address and forwards it to the final destination. This can be done, since the packet is tunnelled and therefore does not has to do any routing. Furthermore the information of the payload is supposed to be sanitised.[2] However, as Tarzan is a network layer, applications may not be aware of any requirements of anonymization in the payload [BG02]. As a result, any application may be tunnelled by Tarzan, but many types of applications may require some plug-in to sanitise the payload.

3.2 Layered Encryption

Within a network where a sophisticated routing within peers is part of the anonymizing features, traceability of data flow should be minimized. So that an adversary cannot discern dependence of an incoming link to an outgoing link merely from the packet content. As well, any kind of content disclosure must be prevented. A complex procedure of layered encryption and integrity checks for each hop is applied in Tarzan.[3] Integrity checks make sure that corrupt data will be dropped and does not cause needless traffic overhead. Layered encryption on the other hand assures that any peer of the routing can only determine the predecessor and successor of the tunnel, but no other participating node. To achieve that, the source of data has to generate l layers of encryption covering the original content, if the path to the tunnel endpoint has length l. Passing the content to the endpoint, every hop takes of one layer of encryption. Thus, after passing $l-1$ relaying peers

[2] As an example: The HTTP-Request-Header usually contains information about for example the language, charset, user-agent of the user, as well as the referrer the query.

[3] Being similar to Chaumian mixes.

(relays) on its way, the endpoint takes of the last layer of encryption and can interpret the data. On its way back through the tunnel every peer adds one layer of encryption, starting with the former endpoint. The original source then has to decrypt the response message l times to see the content. By this design, the actual workload for de- and encryption is placed on the node seeking anonymity.

To illustrate the course of encryption for only one step of the tunnel the following calculation is carried out. Given B_{l+1} being the data packet to be forwarded by node h_l to the final destination h_{l+1}. The source then encrypts the message with the adequate encryption key ek of node h_l: $c_l = ENC(ek_{h_l}, \{B_{l+1}\})$. Then a sequence number is added to c_l, which was set to zero at tunnel setup, and the integrity key ik is applied to calculate the fingerprint a_l: $a_l = MAC(ik_{h_l}, \{seq, c_l\})$. A sequence number seq, fingerprint c_l and encrypted message a_l taken together give the Block B_l: $B_l = \{seq, c_l, a_l\}$. Iterating this calculation $l-1$ - times with the adequate keys will result in the information Block B_1, which can be forwarded to the first relay of the tunnel.

However, to make calculations for de- and encryption efficient, data exclusively is encrypted symmetrically. Moreover, different keys are used for each relay and direction of data flow, for either integrity key and symmetric encryption key. Consequently keys must be exchanged between the source and every relay separately at tunnel setup. To be able to identify a data stream and respectively apply the correct keys on the data, so called 'flow identifiers' are given to every relay at tunnel setup. These flow identifiers uniquely identify each link of each tunnel. Further details on the tunnel setup will be given in subsection 3.5. As well, we will see in subsection 3.4, that an additional layer of symmetric encryption is applied between every set of two following nodes of the tunnel.

We have now seen, how data is encrypted throughout the tunnel. In the further subsections peer discovery, routing within peers of the network and the tunnel setup is described.

3.3 Peer discovery

When a new peer enters the network, it first locally generates itself a public key, to give other peers the possibility to contact it in a secure manner. As a new node initially only knows a small number of other nodes,[4] it then contacts these nodes to learn about all the other nodes. To enable peer discovery in a decentralized but verifiable manner a simple gossip-based protocol is used to distribute an information tupel. This tupel contains the node's IP address, port number and a fingerprint of their public key: $\{ipaddr, port, hash(pubkey)\}$.[5] The ultimate goal is to learn about all network resources. The complete knowledge of the network is necessary to later find mimic partners in the network, as we will see in subsection 3.4.

The peer discovery protocol supports three means of dialogue between two peers, initialization, redirection and maintenance. The 'initialization' permits the transfer of complete neighbour list to a new peer. Where a neighbour list is nothing more than the list of all known peers in the network. Therefore a new peer randomly contacts one of it's initially known nodes. As Tarzan peers distinguish between unvalidated addresses U_a and validated addresses V_a, now every learned IP address is verified by a simple discovery request with a random nonce. As soon as it receives a response message the queried IP will be shifted to V_a. Only validated IP's are part of the IP table used for mimic selection and tunnel setup, as will be explained in detail in subsection 3.4 and 3.5. Both discovery requests and responses are sent directly to the corresponding IP address.

Further 'redirection' gives a queried peer the possibility to shed it's load and redirect the request of a new node to another randomly chosen node.

Finally the 'maintenance' part of the protocol only provides new information to the database of the querying node.As a result, before exchanging IP address information, differences of both the node's IP tables must be calculated. This is done by a k-ary structure[6] on prefix-aggregated hashes of the set elements. Searching for a single difference in this structure can be carried out in approximately $log_k n$ steps. At first only the upper-most hash value of the structure, combining the

[4] How many nodes exactly and where from a nodes knows these few nodes is not further specified in the paper.
[5] No further specifications about the gossip-protocol are given in [FM02a].
[6] Not just a binary one

whole IP address table, is exchanged. Comparison continues if this value is not the same. Peers then exchange $k-1$ hash values at once for every further step.

The network consequently learns and validates the peers in the network in $O(n)$ connections. Counting every single message about c messages for initialization, $2n$ messages for discovery request and response and some $2 \cdot b \cdot log_k n$ messages for maintenance communications should be enough. Summing up to approximately $2n + 2b \cdot log_k n + c$ messages, with b and c probably being smaller than ten. In [MOP+04] the authors of the anonymizing overlay network *AP3* evaluate the peer discovery as not been adaptable enough to scale well in large networks, due to it's dependency on gossiping and a complete IP-table. Especially as Peer-to-Peer networks are known for their rapid changes in topology, they prefer a system, whose routing is based on chance.

Obviously peer discovery can be influenced in a bad way if all nodes initially known by a new node are malicious. By only communicating the set of other malicious nodes as their complete IP address table and not forwarding any gossiped information the new node will only learn and validate malicious nodes, consequently choosing them as their mimic nodes.

3.4 Mimic Selection

As data packets pass from one hop to another, traffic patterns can be analyzed by a widespread eavesdropper . As a solution for this problem Freedman and Morris utilise cover traffic, also called mimic traffic, which flows on a constant basis between participating peers. Therefore, upon joining, every peer establishes connection to k mimics from his set of validated nodes, to then exchange cover traffic with them. At the same time approximately k peers will determine the joining nodes as their own mimic. The *lookup-function* for finding their mimic partners is hash-based and determined by the Tarzan network, as well as the number k of mimic partners to contact for each peer. All together, every peer has an expected number of $E[K] = 2k$ of peers as its mimic partners. Where K is the sum of incoming and outgoing mimic connections.

To establish a bidirectional, time-invariant packet stream between mimic partners, a symmetric key for link encoding is exchanged. As a result, nobody but the two mimic partners can distinguish cover traffic from real data, which now can be injected in the packet stream, thereby substituting the dummy traffic. As well we may annotate, that an additional decryption and encryption calculation must be carried out, on top of the one of the tunnel.

Mimic selection utilizes a hash-based lookup-function to avoid a malicious influence of a global adversary. Still, if hashing were conducted within a single round of hashing from the set of all nodes, queuing up in a single chord ring, a capable adversary could inject an arbitrary number of peers into the network from only a small number of domains, of even just a single one. Assuming an adversary wants to place a malicious peer, so that a specific peer of the network has to select this malicious peer as its mimic partner. This could be realised using a brute-force method to insert a malicious peer at the right position of the hash table.

To minimize this kind of misuse, Tarzan uses a three-level hierarchy dynamic hash table (DHT) in the form of a Chord Ring,[7], whereby every round determines a smaller subnet to select the mimic from. Starting with the first round, the first 16 bits of all domains are taken into account for hashing. Consequently any /16 subnet maps to one position on the hash ring. One of those /16 subnets gets selected and then, the first 24 bits of all domains within this subnet are mapped onto a new Chord Ring for the second round of hashing. In the third and final round, the actual mimic partner is determined from the remaining set of peers of the chosen /24 subnet.

The selection in each round, including the actual peer selection of the last round, is realized by a hash value of the current date and the IP address of the searching peer. For example, the hash value of the first round of node a's first mimic is $hash(a.ipaddr/16, date)$. The /16 subnet chosen for the next round is the smallest id on the chord ring, which is just bigger than, or equal to, the calculated hash value: $id_{16} = min\{id | id \geq hash(a.ipaddr/16, date)\}$.[8]. For the following two rounds, /24 and /32 are applied respectively, and the first mimic partner is determined. As every peer has to find k mimics, a small variation of the lookup-function is used within the same three chord rings. To find the /16 subnet of the i^{th} mimic the hash-function

[7] Detailed information about DHTs and Chord can be found in [SM07].

[8] Selecting the successor of a calculated hash value is a common procedure in DHTs [SM07]

is applied i times during calculation: $\underbrace{hash(...hash}_{i-times}(a.ipaddr/16, date)...)$. Summarizing the three steps of one lookup-function the required query to calculate the i^{th} mimic of node a is defined as: $lookup^i(a.ipaddr) = M_i^a =: b$

Having determined its' mimic partners, the mimic connection to the peer must now be established. To achieve this, a mimic request is sent, whose correctness will be verified by the queried peer before connection establishment. The reason is, that otherwise, any malicious peer could ask an arbitrary peer to be it's mimic, giving far more dynamic and possibilities to the adversary. Also, DoS attacks within the network would be trivial and three hierarchy hashing would make no sense. Therefore a mimic request from node a to node b contains the IP address of a, as well as the mimic-number: $\{a.ipaddr, i\}$ with $i \in \{1, ..., k\}$. Peer b will only accept the request, if firstly the number i is within the range k determined by Tarzan. So an adversary cannot have more than k mimics, even if he configured his own Tarzan implementation. Secondly, node b must come to the same result of mimic selection performing the calculation that node a did, but using his own hash table: $b.lookup^i(a.ipaddr) = M_i^a =: b$. By performing this, node b makes sure, that node a did not choose him specifically.

Two other 'honest' situations are possible where the lookup-check fails. In the first case, a and b may simply have a different view of the network. That means, that b found some node c, being closer to a's lookup hash value than his own identifier on the chord ring. Consequently the IP table of a and b are not identical. Hence, b will refuse the mimic request and send the required information of node c to a. Peer a then contacts peer c to be it's mimic. In the second case, peer a already tried to contact peer c, which did not respond. Therefore, a removed node c from his IP table and his lookup now mapped to node b. In this case, after being refused by node b, node a then sends a new message to node b including that c did not respond. Node b will then ping node c to verify and, if c does not respond, accept a's mimic request.

To conclude, in this subsection we have seen how Tarzan attempts to hinder an adversary from being able to flood the network with numerous malicious peers from just a few domains, and distribute them equally as potential mimic partners. Using a single hash ring, an adversary requires an expected n attempts to place a malicious node into any specific part of the hash table. This is in contrast to the proposed three-level hierarchy hashing, where an adversary cannot 'conquer' most parts of the hash table, as he most probably cannot spoof all /16 and /24 subnets. As a result, if an adversary wants to attack a specific peer, he will find it hard to inject a peer into the network, so that it results in being a mimic of the attacked peer. Thus, much more effort must be taken, than just brute-forcing it's own capacity of IP addresses (approximately n times) to generate a set of mimics for this peer.

3.5 Tunnel Setup

In the previous subsection we have seen, how every participating peer selects its' mimics from the set of all available peers of the network in a predetermined way. This selection algorithm is hard to influence if just one of the corresponding nodes is honest. On the other hand, if both peers are malicious they can easily configure the protocol and accept an arbitrary mimic request. In this subsection we will see how the tunnel setup algorithm safeguards an honest node from inadvertently rooting through a tunnel of self-chained malicious nodes.

Assuming initiator node a wants to setup a tunnel $T = (h_1, ..., h_l, h_{pnat})$ of length $l+1$. Other than in Onion Routing, described in [MOP+04], node a does not know which nodes to pass during routing. To possibly achieve this, node a at tunnel setup would have to know the mimic partners from every peer in the network, to predetermine the path. Many hash calculations would be required to fulfil this time consuming task. Unless constantly actualizing, node a would not have the required amount of time, especially if a tunnel fails and would have to be reconstructed (see subsection 3.6). Obviously in both cases the network would scale very poorly. In contrast, Tarzan successively sets up the tunnel, starting with one of his own mimic partners. To initiate the tunnel setup, node a chooses node h_1 at random from its' mimics $\{M_1^a, ..., M_K^a\}$ and injects a query data packet, addressed to h_1, into cover traffic. This packet is encrypted by the public key of h_1 and the common link encoding of the mimic partners and contains a session key only. Node h_1 responds

with it's mimic list $\{M_1^{h_1}, ..., M_K^{h_1}\}$. As initiator a can perform the same lookup in the chord ring as node h_1, a then verifies this list and chooses the next hop of the tunnel at random, if verification succeeds. The same verification is done for the expected k peers that contacted h_1 to be their mimic, as they are also mimic partners of h_1.

Verification of self-chosen mimics: $a.lookup^i(h_1.ipaddr) = M_i^{h_1} \quad \forall i \in \{1, ..., k\}$

Given that verification must be successful, a node cannot have arbitrarily chosen nodes as its mimic partners and involve them in the tunnel. So once again, the complete view of the network is very important for both nodes.

Having chosen and contacted the next node h_2 of the tunnel, the initiator sends a data packet to node h_1 containing the forward integrity key, the subsequent reverse keys for packets from h_2, the IP addresses of a and h_2, and their flow identifiers. By these flow identifiers, node h_1 can now determine the corresponding keys for each data stream and might build up a table as in figure 2, where for every tunnel relay two entries are made, one for each direction:

No.	Identifier	Corresponding Keys + Destination
1.	{IPaddr1,flowID1}	{integrityKey,IPaddr2,flowID2,symmetricKey}
2.	{IPaddr2,flowID2}	{integrityKey2,IPaddr1,flowID1,symmetricKey2}
..		

Figure 2: Hypothetical table entries of a relaying peer, with two entries for each tunnel relayed. $Key2$ stands for the reverse keys.

By now, initiator a can tunnel any data through node h_1 to h_2, without h_1 being able to distinguish successive establishment requests from ordinary tunnelled data. This process is repeated l more times until reaching the end of the tunnel h_{pnat}. The only exception in this process is, that the last node h_{pnat} is not chosen from the mimic partners of node h_l, but at random: $a.lookup(random)$. Thus, data traffic from h_l to h_{pnat} is encrypted, but visible to a global eavesdropper. The reason for this process is, that if the tunnel connection fails, the connection to h_{pnat} can be set up again from any other predecessor node h_l'. As a result, the connection to the actual server is continued by the same node h_{pnat} and higher-level applications won't die upon tunnel failures and reconstructions. Further details are given in subsection 3.6.

Coming to the complexity of the tunnel setup, Freedman and Morris point out, that $O(length)$ public-key operations and $O(length^2)$ inter-relay messages are exchanged. They therefore consider the overhead as being sufficiently small for realistic choices of the tunnel-length. The authors point out that for tunnel setup an expected $20ms$ are required for each hop. The de- and encryption process merely delays the routing for another $1ms$ at each hop. This is less than ordinary routing times. In contrast, in [MOP+04] Tarzan is considered to have a significant overhead, due to key exchanges and encryption. However, no detailed considerations for this assumption are given. Thus, the exact time values stated, seem to be more convincing.

One question is, how the length of the tunnel should be determined. As discussed in the appendix of [FM02a], Tarzan has a variable path length. Although, neither a detailed function is given for calculations nor, it is said, that the length is chosen at random. As a comparison, in Crowds [RR98] peers decide by flipping a coin, if the message is forwarded once more or sent to the destination. Both possibilities occur with a probability of 50%, so a successor would not be able to draw any further conclusions if, for example, he knew the number of peers following him on the tunnel. The probability that the predecessor is not the initiator will always be 50%. A similar approach in Tarzan is possible, especially when combined with starting with a fix path length to gain some minimal tunnel length. This can be done, in contrast to Crowds, as relay nodes would not be aware of any probability decision carried out by the initiator at tunnel setup.

3.6 Tunnel Failure and Reconstruction

To constantly check the tunnels functionality, the initiator nodes regularly sends a ping message to the PNAT. Upon multiple unsuccessful pings, the initiator then starts to ping each relay. If h_i is the last relay to respond, the initiator will suspect node h_{i+1} of being the point of failure. Now there are two cases, which determine which type of reconstruction has to be applied.

The first case would be, if only the PNAT is unreachable and all other nodes respond. In this case only h_{pnat} needs to be replaced, which will be connected to the penultimate node h_l and determined by a new $a.lookup(random)$. As h_{pnat} previously contacted the actual destination with his IP address, most connections of the application layer will die on the replacement of the PNAT.

The second and more probable case is, that any of the relay nodes stopped forwarding packets and must subsequently be replaced. Unfortunately, the tunnel is constrained to the mimic infrastructure and there probably is not another node in the tunnel having the same mimic partner as h_i. Consequently, all nodes following h_i must be replaced, except for h_{pnat}. The tunnel reconstruction function accordingly has to reuse the tunnel setup function starting from h_i, building up a tunnel as follows: $T' = (h_1, \dots, h_i, h'_{i+1}, \dots, h'_l, h_{pnat})$. If for node h_i numerous attempts at reconstruction fail, i will be decremented by one and the process repeated until ending successfully. Of course the delays for pinging and reconstruction might exceed the restrictions of a higher-level protocol causing them to terminate. However, most of the time this should not be the case.

Introducing a node h_{pnat} at the end of the tunnel and offering a tunnel reconstruction function are both concessions to the tunnel setup for efficiency reasons. Principally Because, as the tunnel fails, they help to rebuild the tunnel a lot faster and eventually without the loss of the ongoing session from the broken tunnel. On the other hand, the algorithm described by Freedman and Morris, offers extensive possibilities to an adversary. Assuming that a substantial part of the network consists of malicious nodes working for the same adversary, or an adversary is capable of introducing nodes into the mimic structure using brute-force methods. The network could then be filled with malicious nodes, which link up to, at least, one other malicious node as their mimic partner. If, under these circumstances, one of the tunnel relays is a malicious node as well, it may be able to influence the tunnel reconstruction protocol in a malicious manner. To achieve this, no special effort is required, as the node must merely stop forwarding traffic of this specific tunnel, which is labelled by two corresponding flow identifiers. The initiator node will think that the successor node is the point of failure and initialize reconstruction starting from the adversary node. With some luck, the initiator will choose the malicious mimic at random, which then continues with the same tactic to make sure that all the following nodes are chosen from malicious mimics as well. Whenever the initiator selects another honest mimic, the adversary node will have to repeat pretending a tunnel failure until successful. Obviously this tactic might take multiple attempts and some time. It is however, extremely simple to implement and will lead to the final node h_{pnat} of the tunnel. Then h_{pnat} can be eavesdropped upon and the final destination disclosed. Additionally the length of the tunnel might lead to some extra information for the adversary. It follows that the disclosure of recipient activity and content by a malicious intermediate node was answered 'NO' in Table 1 of [FM02a] and might not be correct from this new point of view.

3.7 Cover Traffic

Two of the most important features to achieve in an anonymizing system, are the deniability of every node being the source of data and the untraceability of data flow within a network. To provide these kind of features, cover traffic is applied in Tarzan, that is supposed to flow between mimic partners on a constant basis. As a result, incoming cover traffic can be dropped or rebalanced at any time if real data has to be sent. Outgoing cover traffic can be created if data is received. Link encryption ensures that cover traffic is indistinguishable from real data. Additionally, no variation for the amount of data exchanged can be perceived.

Still, some traffic invariants will have to make sure, that no node can be identified as being a clear source or sink of data. For this purpose Freedman and Morris introduce two equations. The most important being an estimation to constrain *the outgoing 'real' data to a single tunnel* to be smaller, or equal to, *one third of the total incoming rate* (meaning data + cover traffic). The estimation is based on the fact, that the restriction '$\frac{1}{3}$' is sufficient, if at least $\frac{1}{3}$ of its' mimics

is honest. The other equation makes sure that a node will not be a clear sink of data and gives possibilities to adjust or raise the amount of cover traffic between peers.

We show that Tarzan imposes minimal overhead over a corresponding non-anonymous overlay route. Latency through Tarzan tunnels is dominated by transmission speed through the Internet.

Consequently, cover traffic may offer a high level of deniability and untraceability. However, it may be criticised for causing a large overhead. Considering the given equations, this overhead must be at least double size of the real data sent. Still one may argue, that if not applying cover traffic, other techniques must be utilised to hinder traffic analysis. As described in [SS03], batching of data packets, and therefore delaying them, in every forwarding peer can be a possible way to minimise traffic analysis. To optimize this effect, concentrating traffic on a decreased number of routing peers is proposed. However, batching data to an extent that minimizes untraceability, will always result in not being applicable when operating in real-time is desired.[9] As a result, this technique is not an alternative to cover traffic. Still, if applied at the PNAT only, it might help to improve security without interfering round-trip times too much. This is because the PNAT probably is the weakest link for traffic analysis attacks.

3.8 Further Possibilities

Some important possibilities deriving from Tarzan, that have not been mentioned so far, are the following. First, as the PNAT in Tarzan also offers port forwarding, ordinary Internet hosts might connect through Tarzan to anonymous servers. This is because the server constantly builds up new tunnels to the same PNAT, to conceal it's identity, but always offering the same service via this PNAT. Points of criticism are, that the PNAT needs to be a hundred percent reliable peer, to always be reachable. Additionally, it appears to be the one responsible for offering the servers information from the perspective of any outside user.

Secondly, both sender and recipient can achieve anonymity at the same time, setting up a so called *double-blinded channel*. Therefore, both users must set up a tunnel to a different PNAT, and then each user's application connects to the other's PNAT by an ordinary Internet connection (see figure 3). As a result, data flow leaves the network and should be encrypted by both users' applications. Points of criticism are, that again, both PNATs used must be very reliable. As well, both users would have to agree on a specific time to start communications, otherwise both would have to listen constantly on their accordant PNAT for a connection setup.

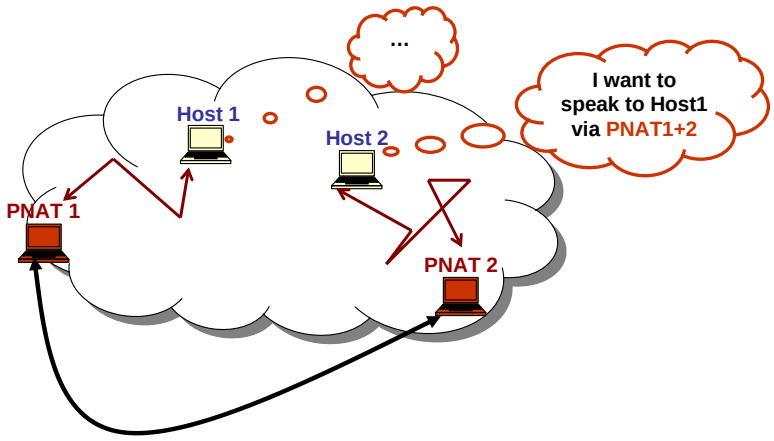

Figure 3: Two Tarzan nodes setting up a double-blinded channel.

Setting up a direct connection through one tunnel, within Tarzans' mimic connections, theoretically is possible as well. However, as routing cannot be foreseen, both peers would have to calculate all mimic partners of all nodes to find a possible routing. As well, the loss of any single

[9] In Tarzan data packets are batched for/in 20ms intervals, due to introducing batches into cover traffic. This is far to less to achieve untraceability [SS03].

relay on that tunnel could not be fixed by a simple tunnel reconstruction protocol, as a completely new path for routing must be found.

Summarizing, it can be said, that both of these possibilities are far from an optimal implementation of this kind of communications.

4 Security Analysis

To examine an anonymizing system's security, a wide range of adversaries must be taken into account. Furthermore, a system cannot be understood to be the best solution for all anonymizing tasks, which might be required in a diverse field of applications and system environments. Specifically a trade off between performance and level of security must be given to determine the type of anonymizing system to apply. This is because higher security, among others, comes with more complex methods of encryption and path selection. Another consideration concerns the adaptability of an anonymizing system to a single application, or offering a general approach for all applications. Tarzan was designed to be independent of applications and to have a higher level of security, while nevertheless retaining interactive TCP communication.

Still, more or less potential attacks and general threats could be pointed out in the previous sections. One of these general threats is the disclosure of information by an adversary through content in the application layer payload, if not sanitised by the application itself. Tarzan does not face this problem. As a result, all applications using Tarzan must be aware of sanitising requirements in order not put others risk others. Therefore, a tool anonymizing one a specific type of application might be more adequate for an unawary user.

In the following subsections potential and less likely type of attacks will be discussed.

4.1 Prevented and Unlikely Attacks

As discussed in detail in the section *Applied Techniques* some known attacks and threats might be prevented or strongly complicated by Tarzan's design. As pointed out by the authors, the following list of attacks given in open-admission, self-organized peer-to-peer models have been faced:

Attacks through corrupt gossiping - As indicated in subsection 3.3 should only be possible to corrupt gossiping, if all initially known peers, of a joining node, are malicious. In that case, the joining node will always keep an incomplete IP-Table, with exclusively malicious peers.

Attacks given by open admission - As discussed by the author, an adversary may control many peers in a number of domains, but not the entire Tarzan network, due to subnet-hierarchy hashes for IP-Tables (see subsection 3.4). As well, public keys are gossiped and not distributed directly. Consequently an adversary cannot respond with a unique public key to each request from a specific peer, building up a table with the peer identity and the unique public key. By doing this an adversary, could identify the initiator of a tunnel setup request.

Attacks per ignoring neighbour-selection algorithm - At tunnel setup, mimics of all relays are verified by the initiator and arbitrarily set mimic connections will not be tolerated. As well, unless being able to spoof all /16 and /24 subnets, an adversary probably cannot 'generate' all mimic partners to a specific peer. Still, a capable adversary might be able to generate some of them, using brute-force methods to break hashing (see subsection 3.5.

Attacks by adaptive, compromising adversary - The authors point out, that Tarzan offers far less possibilities to an adaptive adversary than most of the other anonymizing systems. Especially, to systems that use a central core network, for example, diverse types of Mixes. In Tarzan, tunnel duration and mimic stability are generally too small for an adversary, that wants to successively compromise peers to finally discover the initiator. As well, different operating systems in most peers make it difficult to rapidly compromise said peers.

Attacks on mimic nodes by sudden mutual omission of cover traffic - If the peer to be attacked, is surrounded by cooperating mimic nodes, they then may, at the same time, omit sending cover traffic and expose the peer as been a source or relay of data. Still, given the invariants in subsection 3.7 they must sum up, to at least $\frac{2}{3}$ of the peers mimic, to do so. This is very unlikely due to the specific type of hashing applied in Tarzan.

Attacks by interpreting content - Due to complex encryption and the integrity mechanisms employed in Tarzan, this type of attack is virtually be impossible, even from within the tunnel. Obviously the data is decrypted as soon as reaching the PNAT, and can be seen by a global eavesdropper or, if further encrypted in payload, at least by the destination webserver.

Attacks through traffic analysis - Given the cover traffic and considering the equations given by the authors (see subsection 3.7, possibilities of traffic analysis are fairly weak for any adversary outside the tunnel. However, any tunnel relay may apply cover traffic techniques to obtain some information. Especially when cooperating with a global eavesdropper, that might be observing the traffic of the PNAT.

4.2 Possible Attacks

Some possible attacks on Tarzan were already discussed in previous subsections. The **Attack on Tunnel Reconstruction Protocol** for example, enables an adversary placed in the tunnel, to trigger and interfere with tunnel reconstruction (see subsection 3.6) A possible improvement will be presented in section 5 . As well some information about an **Attack via information of application layer** was given in sections 2 and 4. Some more attacks will now be presented, starting with the most promising one.

The **Intersection attack** is a passive logging attack [Ray01]. It is possibly the most powerful of all attacks, while remaining extremely easy to implement. More networks than just Tarzan are affected and it is stated, that relavively few means of defence exist [WALS03]. Tarzan is one of the vulnerable system primarily because the peers communicate directly overThe main aspect making Tarzan such an easy target is, that only a single peer in the Tarzan network is required to obtain the complete IP table of all nodes in a passive manner. Whenever a user contacts a specific webserver and can be distinguished from other potential users, a copy of the actual IP-address table is saved. Taking a collection of timely disjoint copies of the IP table, specifically when the was 'online', helps to decrease the list of possible IP's. Therefore the different IP tables only need to be cross-referenced. This may finally reveal the user's IP, or shrink the IP table to a number which could be observed with other eavesdropping techniques, to finally be identified. This attack is extremely efficient even for low bandwidth protocols like SMTP, as frequency of connection is low and therefore the table of IP's varies a lot.

In [Ray01] the uptime of a user can best be modelled by a pareto distribution. As this distribution is known for it's heavy tail property, it becomes clear, that most other peers will leave the network after a short amount of time. Some however, stay indefinitely. The authors therefore show, that fully exposing a user is fairly unlikely in a realistic amount of time. Still, reducing the set of IP's to a number of ten or less, can be achieved within two weeks time at a probability of 0.999 percent. For Tarzan this might even be too pessimistic from the adversary's point of view. As Tarzan is real-time applicable, the average uptime of a user is generally not longer than an hour, so the frequency of saving the IP table can be much shorter than an hourly rate, consequently reaching a small set of peers much faster.

Attacks by sending data via suspicious node - Adversary nodes may set up a tunnel via a suspicious node to then send data through the tunnel. Given the equations from subsection 3.7, the adversaries can calculate the maximum amount of data that can pass through this node. Occupying exactly this amount of data for their traffic, the loss of data might be calculated, resulting in a fair estimation of other traffic passing or leaving this node. To lessen side effects of other nodes, at least one adversary should be the mimic of the suspicious node.

Attacks by traffic and timing analysis - Traffic analysis including timing analysis attacks can be realised by any relay node of the tunnel, as well as by a global adversary to the tunnel nodes h_l, h_{pnat} and the destination. If both can be combined, a tunnel node may be able to guess, to which PNAT his relayed information flows. If timing analysis works well, even the distance to the tunnel end might be estimated. As well, stopping forwarding traffic for a short amount of time, by a malicious relay node, may help to verify the accordant node at the end of the tunnel.

5 Improvements

Most other network developers, that criticise Tarzan, could not propose any possible improvements as use a different design. In the following, I will list some improvements to the system that I find to be applicable.

- A slight variation of tunnel reconstruction protocol may be able to avoid the influence of an adversary. Therefore, reconstruction has to start at node h_{i-1}, if node h_{i+1} fails. Node h_i accordingly cannot gain any advantage, as it is no longer the start-point of tunnel reconstruction.
- Further batching of packets at the PNAT node may lessen the possibility of 'linkability' of node h_l to the final destination. As well concentrating the PNAT connection to a subset of peers can enhance 'unlinkability'. However, Tarzan would therefore have to give up its' fully peer-to-peer design.
- Nothing was said in the [FM02a] about having more than one tunnel. I find connections to be more reliable and traffic analysis harder if every user builds up more than just one tunnel to the same PNAT. Consequently employing two or more tunnels for one higher-level connection. For other higher-level connections, for example other applications, a tunnel to a different PNAT seems beneficial. This might hinder the user-linkability of applications and minimize the discussed weakness of Tarzan (see section 4).
- Last but not least, if proxies are employ and changed on a regular basis, intersection attack is less effective. Further, the same user can exclude himself from the set of potential IP's by masquerading with different IP addresses. Obviously this is not feasible for every user and is not an improvement of Tarzan itself.

6 Conclusion

Tarzan is a fully peer-to-peer approach of a network layer that provides anonymity that may be used for real-time applications. The open admission though, requires some means to protect users from a global, adaptable adversary. Therefore, a more sophisticated hashing has to be applied, that determines the mimic selection and tunnel setup. As a result even brute-force methods can only influence 'some' parts of the network. Gossiping of peer information and validation help to arrange peer discovery. Consequently peer discovery cannot be influenced, except if all initially known nodes of a joining peer are malicious. To enable deniability, cover traffic and encryption are employed within the network. However, as encryption ends at the last node, it is important that application layer payload is sanitised. Users and applications may not be aware of this fact and put themselves at risk. Cover traffic ends at the penultimate node of the tunnel, therefore the last two nodes can be held responsible for the content relaying to the initiator. In case of a malicious peer being on the tunnel, further traffic analysis is possible and can disclose more relaying nodes of the tunnel. Still, definitively identifying the initiator cannot be guaranteed, unless encircled by adversary mimic nodes. However, the more influential the adversary gets, the higher the probability is of a specific node being the initiator. Depending on the further capabilities of the adversary, isolating a small group of peers might be enough to, in the following, expose the initiator. Some improvements on the network listed in section 5 might help to minimise some of these kind of attacks. Probably the most significant weakness of Tarzan is based on the knowledge of every peer of all IP addresses of the network, which is necessary to employ mimic selection and tunnel setup. As a result, passive logging attacks can be employed by a single malicious node cooperating with a global eavesdropper or specific webserver. Unless changing their own IP address at frequent intervals, this can result in a rapidly decreasing set of potential initiators. In short, responsibility of the PNAT node, possible passive logging attacks and a lack of interested users has resulted in Tarzan, nowadays, not being utilised as an anonymizing network layer to a large extent. However, interesting ideas have been included, that may have strongly influenced posterior approaches.

References

[BG02] Krista Bennett and Christian Grothoff. GAP — practical anonymous networking, 2002.

[FM02a] Michael J. Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, Washington, D.C., November 2002.

[FM02b] Michael J. Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer (slides) - `http://www.scs.stanford.edu/mfreed/docs/tarzan-ccs02-slides.pdf`, 2002.

[MOP+04] Alan Mislove, Gaurav Oberoi, Ansley Post, Charles Reis, Peter Druschel, and Dan S. Wallach. Ap3: Cooperative, decentralized anonymous communication, 2004.

[Ray01] Jean-François Raymond. Traffic analysis: Protocols, attacks, design issues and open problems. In H. Federrath, editor, *Designing Privacy Enhancing Technologies: Proceedings of International Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *LNCS*, pages 10–29. Springer-Verlag, 2001.

[RR98] Michael K. Reiter and Aviel D. Rubin. Crowds: anonymity for Web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[SM07] Christian Schindelhauer and Peter Mahlmann. *P2P Netzwerke: Algorithmen Und Methoden*. Springer-Verlag, March 2007.

[SS03] Andrei Serjantov and Peter Sewell. Passive attack analysis for connection-based anonymity systems, April 2003.

[WALS03] Matthew Wright, Micah Adler, Brian N. Levine, and Clay Shields. Defending anonymous communication against passive logging attacks, 2003.