Covert-channel-resistant Congestion Control for Traffic Normalization in Uncontrolled Networks

Martin Byrenheid
Technische Universität Dresden
Privacy and IT Security
martin.byrenheid@tu-dresden.de

Michael Rossberg and Guenter Schaefer
Technische Universität Ilmenau
Telematics and Computer Networks
{michael.rossberg,guenter.schaefer}@tu-ilmenau.de

Robert Dorn
secunet Security Networks AG
robert.dorn@secunet.com

***Abstract*—Traffic normalization, i.e. enforcing a constant stream of fixed-length packets, is a well-known measure to completely prevent attacks based on traffic analysis. In simple configurations, the enforced traffic rate can be statically configured by a human operator, but in large virtual private networks (VPNs) the traffic pattern of many connections may need to be adjusted whenever the overlay topology or the transport capacity of the underlying infrastructure changes.**
We propose a rate-based congestion control mechanism for automatic adjustment of traffic patterns that does not leak any information about the actual communication. Overly strong rate throttling in response to packet loss is avoided, as the control mechanism does not change the sending rate immediately when a packet loss was detected. Instead, an estimate of the current packet loss rate is obtained and the sending rate is adjusted proportionally. We evaluate our control scheme based on a measurement study in a local network testbed. The results indicate that the proposed approach avoids network congestion, enables protected TCP flows to achieve an increased goodput, and yet ensures appropriate traffic flow confidentiality.

I. Introduction

As large companies and government agencies increasingly rely on the computer networks to transfer confidential information, they also become more attractive targets for espionage attacks. Recent revelations show that intelligence agencies perform traffic analysis on selected targets even on a global scale [1].

While most state-of-the-art methods in network security concentrate on the confidentiality and integrity of message contents, they leave protection against attacks based on pure traffic analysis (so-called traffic flow confidentiality, TFC) out of scope. Yet, studies, like [2], show that an eavesdropper may still be able to identify communicating parties and type of exchanged content by simply learning about traffic patterns of encrypted packets. In case of a more elaborate attack, e.g. by deploying a compromised keyboard [3] or Trojan Horse in a virtual private network (VPN), an attacker may even use intentionally generated traffic patterns to bypass mandatory encryption and leak information to a colluding eavesdropper in the open Internet, thus realizing a covert channel.

Due to their high significance in high security scenarios, we will concentrate on the context of VPNs within this paper. Fig. 1 shows an example for such an attack in a VPN, where a compromised node (labeled *insider*) in a protected network area A systematically sends packets either to a node in area B or C to transmit 0 or 1, respectively. The eavesdropper then

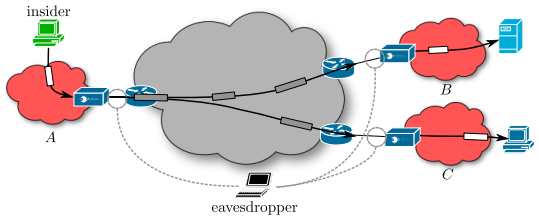

Figure 1. Scenario: espionage attack on a VPN utilizing covert channels

infers the data from the destination address of each packet. However, the destination address is just one of several covert channels that have been found in IP networks. For a taxonomy of currently known covert channels in computer networks, the reader is referred to the recent work by Wendzel et al. [4].

A frequently proposed countermeasure to mitigate covert channels is the automated detection of their exploitation [5], [6], [7], [8]. Although detection seems attractive because of its comparatively low overhead, it has three major drawbacks: First, it cannot prevent covert channel attacks completely, as a certain number of packets need to be observed before an attack can be detected reliably. Second, no reasonable upper bound on the bandwidth of the remaining undetectable covert channel can be proven. Third, an automated detection offers no protection against passive observation of patterns in legitimate traffic, i.e. TFC in scenarios without an insider.

Achieving perfect resilience against traffic analysis requires normalizing all data traffic by padding packets to equal length, creating cover traffic and normalizing header fields. This perfect approach is often considered impractical due to its high overhead. However, it is possible to reduce the overhead in a controlled manner by utilizing *mode security* [9], which still allows to obtain tight upper bounds for potentially leaked information.

However, no matter if mode security was enabled or not, a very practical problem remains: When normalized traffic flows are sent over dynamic networks, such as the Internet, it must be possible to adjust the traffic pattern at runtime in order to react to changed network conditions. Without loss of generality, we consider multiple gateways forming a VPN. Every pair of gateways sends encrypted packets with fixed length and normalized header fields at a fixed time interval, generating dummy packets when necessary. The length of the packets as

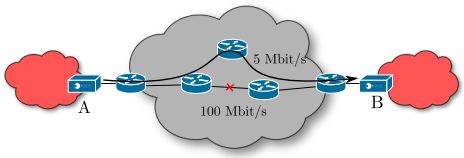

Figure 2. Example where a link failure leads to a decrease of the bandwidth from 100 Mbit/s to as few as 5 Mbit/s: If the consumed bandwidth of gateway A is higher than 5 Mbit/s, the bottleneck link will be overloaded.

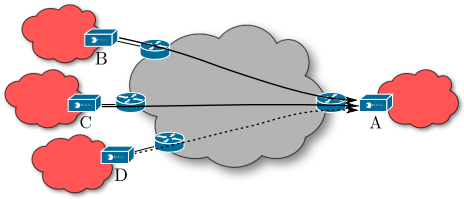

Figure 3. Example illustrating a major scalability problem due to static traffic normalization: If node D establishes a connection with node A, the bandwidth consumed by node B and C must be reduced to avoid congestion at the uplink of node A.

well as the transmission interval define the *consumed network bandwidth*. When such a network is set up, one must consequently select a consumed bandwidth that meets throughput demands while also avoiding network congestion. In dynamic networks however, selecting the consumed bandwidth *a priori* in a manner that congestion is entirely avoided is impossible in general. Fig. 2 and Fig. 3 show examples for situations where congestion may occur in our setting.

The consequences of congestion are a higher latency, more jitter and packet drops. The latter in turn will throttle TCP connections, including those transported *within* the VPN-protected stream. In traditional VPNs without traffic normalization, the automatic throttling of the protected TCP streams in response to packet loss will also reduce the traffic sent between the gateways, therefore avoiding further congestion. If traffic normalization is applied however, any throttling of the protected flows will only result in a proportional amount of additional cover traffic being generated in order to conceal the pattern of the protected traffic. Therefore, the congestion will not vanish, and a severe decrease of TCP efficiency will be a direct consequence. Furthermore, due to the higher packet loss, latency and jitter caused by an appropriate amount of cover traffic, also real-time traffic like VoIP connections will be negatively affected.

To the best of our knowledge there is no research examining this condition, yet. Thus, we contribute a first effective mechanism to perform network congestion control, without the possibility of leaking information. This is an important prerequisite for ensuring traffic-flow-confidentiality in Internet-like networking scenarios. In a measurement study we evaluated a prototype implementation under reproducible conditions within a testbed.

The rest of the paper is structured as follows: the following section will give details on objectives, adversary model and related work (Sec. II), before Sec. IV describes our novel approach to congestion avoidance. Sec. V discusses the evaluation. The paper closes with a conclusion and outlook.

II. System Objectives & Adversary Model

A traffic-flow-confidentiality-aware scheme to control the bandwidth consumed by VPNs, should meet the following objectives:

- **Utilization of available bandwidth:** While inappropriate bandwidth consumption leading to congestion must be avoided, restricting capacity if there is still network bandwidth available may also be undesirable. Thus, the mechanism shall utilize the available bandwidth as best as possible, thereby maximizing the goodput of protected TCP connections.
- **Responsiveness:** VPN-protected TCP traffic will quickly be rate-controlled in response to packet loss. As it is impossible to modify the TCP behavior all client devices, the scheme shall react to network congestion timely.
- **Avoid information leakage:** The scheme must not reveal information about the actual traffic between protected networks. Consequently, adaptation needs to be done solely based on characteristics of the public network.

All in all, there is the technical need to utilize the available bandwidth as best as possible, while quickly reacting to changes. At the same time there must be no influence of clients in protected networks on the rate of the encrypted flow, as this would allow insiders to establish covert channels.

Adversary Model

Like pointed out already, we consider a federation of network sites being connected over a shared infrastructure, such as the Internet. Each site is protected by a VPN node that acts as gateway to the other sites and uses encryption and traffic normalization to conceal actual data transfers. The VPN nodes are the only devices that have access to the shared infrastructure and every VPN node only supports transmitting packets to other VPN nodes.

As already indicated by Fig. 1, we consider a strong form of attacks on traffic flow confidentiality, i.e. by a combination of an insider and an eavesdropper. The insider has access to confidential information that it wants to transmit to the eavesdropper by sending IP packets with noticeable traffic patterns. The insider may be able to corrupt every other node in the protected network, except the VPN gateways.

To consider the worst case, we assume the eavesdropper to be able to directly observe the interfaces that connect each VPN gateway to the shared network. Due to the model, the eavesdropper is unaffected by network dynamics such as jitter or packet loss. However, we require the eavesdropper to be computationally bounded in the sense that it cannot decrypt any packet sent from a VPN gateway. While the rest of the paper assumes each protected network to consist of one or more physical devices, our model is also applicable to so-called *road warriors* having a single VPN-capable device which runs multiple (untrustworthy) processes.

III. Related Work

In the following, we start with a short survey of existing countermeasures against traffic analysis and their adaptiveness to network congestion. Subsequently, we discuss potentially relevant congestion control schemes.

A. Congestion-Aware Countermeasures against Traffic Analysis

While many countermeasures against covert channels in computer networks aim to provide a more or less successful automatic detection of exploitation, several preventive measures have also been proposed. The NetCamo-system [10] combines traffic normalization with a central management entity. In order to choose an optimal bandwidth consumption for each pair of nodes, it is assumed that the central entity always knows the available capacity of all network links over which traffic can be sent. Based on this knowledge, NetCamo is able to prevent traffic analysis while avoiding any congestion at the same time. In large shared and dynamic networks however, the available bandwidth of network links is typically unknown and also subject to continuous change, rendering NetCamo unsuitable for our context. Furthermore, it introduces a single-point-of-failure.

Another solution that explicitly incorporates congestion control is the *network pump* [11]. The approach was designed for multi-level secure networks, where data packets are only allowed to flow from lower level processes to higher level processes. To enable reliable communication, high level processes send acknowledgements to the corresponding lower level processes. Since a compromised high level process could systematically time its responses to leak information, the network pump acts as a mediator between both levels. It ensures that at least a certain amount of time has passed between two consecutive acknowledgements to the same lower level process. Due to the increasing delay when the network gets congested, the network pump automatically prevents congestion collapse. Although the basic idea could be adapted to non-multilevel networks, it is not considered to be suitable, as it does not completely prevent information leakage.

More recent is Sadeghi et. al's work [12], which proposes a combination of traffic normalization and mode security, where each mode corresponds to a specific bandwidth consumption, e.g. 1Mbit/s, 5Mbit/s and 20Mbit/s. Whenever a VPN gateway initiates a mode transition, the new mode is chosen according to the amount of traffic coming from the associated protected network. An implicit adaptation to congestion can therefore be achieved, if the internal traffic demand automatically decreases due to high latency or packet loss, as then a mode with lower bandwidth consumption will be chosen during the next transition. Consequently, the quality of congestion control highly depends on the granularity of mode transitions, e.g. increase/decrease in steps of 0.1Mbit/s vs. steps of 1Mbit/s, and the transition interval, e.g. every 10 milliseconds vs. every second. While a higher granularity and a lower transition interval enable a better adaptation to congestion, they also increase the exploitable covert channel bandwidth. Hence, this approach inherently cannot achieve a good utilization of available bandwidth and prevent leakage of information at the same time.

B. Congestion Control without Guaranteed Delivery

We aim for a protocol that provides congestion control, but does not guarantee delivery – in contrast to TCP. Many unreliable congestion control protocols [13] have already been developed in the context of multimedia streaming applications. One primary focus of these congestion control protocols is to achieve TCP-friendly behavior, which means that the protocol's long-term throughput does not exceed the throughput of a TCP flow under the same conditions [14]. However, selecting TCP-friendliness to be a primary goal is rather problematic in the VPN context. Consider a VPN scenario with five simultaneous TCP transmissions being tunneled through one VPN connection. In a traditional VPN without traffic normalization, the encrypted traffic from the gateway would then also automatically consume the bandwidth of five TCP transmissions. If a TCP-friendly traffic normalization is taking place instead, the encrypted traffic from the gateway would consume the bandwidth of just one TCP flow, hence slowing down the protected TCP flows. Adjusting the consumed bandwidth proportionally to the number of currently running TCP flows would leak information about the protected traffic. Consequently, full TCP-friendliness, i.e. similar short term behavior, cannot be realized under the given goals, rendering existing protocols unsuitable.

IV. Congestion-Aware Traffic Normalization

To prevent leakage of information, rate adaptation must be executed such that it is fully independent of the current rate of tunneled traffic. Therefore, we propose to perform rate adaptation solely based on the conditions in the public network by assessing visible traffic patterns. Any data being exchanged to control the rate is transmitted in protocol fields that are clearly separated from any transported data.

The following section discusses how feedback is transmitted in our protocol, first. We then continue by introducing a basic communication model for two VPN nodes. In Sec. IV-C, we give details on our congestion control scheme.

A. Transmission of Feedback

In difference to usual network scenarios, the targeted bandwidth is always fully utilized in the considered VPNs. Thus, there is a significant and constant share of bi-directional traffic flowing between the devices. Already for efficiency reasons it makes sense to piggyback any control information in the exchanged data packets. Such an approach has an additional advantage: the control information may easily be protected by the cryptographic routines of the VPN, i.e., potential attackers may not perform subtle denial-of-service attacks by tampering with the control information. To achieve the same resilience, explicit feedback packets would need to be padded to the size of data packets, introducing significant overhead and could potentially lead to congestion themselves.

Hence, we propose that VPN nodes embed congestion feedback into any packet sent, including dummy packets. This approach simplifies congestion control and cryptographic routines because every packet may be handled in the same way. It also guarantees that congestion control feedback can be delivered as long as there is no complete interruption of communication.

A potential negative aspect of this mandatory piggybacking is the slight reduction of the maximum transmission unit (MTU) for client devices, even if there is no congestion at all. In comparison to the overhead introduced by VPN protocols, however, this is negligible. Like already pointed out, the alternative of explicit acknowledgments would lead to an even higher overhead in total.

B. A Simple Model for Bandwidth Control

Fundamentally, the consumed network bandwidth is defined by the length of the transmitted packets and the transmission interval. Thus, in theory, congestion control can be exercised by changing either packet lengths or transmission interval. Like in other IP networks, modifying the packet lengths has the major drawback that it requires additional measures to cope with the situation, i.e., so that packets exceeding the variable MTU may still be tunneled. This includes techniques like fragmentation, Path MTU discovery, and TCP proxying, all having a major impact on performance and implementation complexity. Therefore, also for VPNs a sole adjustment of the transmission interval is favorable.

Based on the outlined feedback loop between every pair of VPN nodes, we model the VPN connection of a node A to a node B at a fixed point in time by the following three variables:

- $S_{goal}^{A \to B}$ represents the *intended sending rate* at which node A should send packets to node B given by an administrative constraint, e.g. a fixed pre-configured value (every 10 milliseconds, for example) or even a dynamic value controlled by an algorithm implementing mode security.
- The *current sending rate* $S_{current}^{A \to B}$ defines the rate at which node A is currently sending packets to node B. It is controlled by the congestion control mechanism.
- The *maximum sending rate* $S_{max}^{A \to B}$ reflects the rate that the communication path from node A to node B is currently able to handle given the fixed length of normalized packets. The exact value of $S_{max}^{A \to B}$ is unknown to the device and assumed to be dynamically changing due to traffic conditions. In particular, this value can be lower than $S_{goal}^{A \to B}$.

C. Rate-Based Congestion Control

One fundamental limitation of window-based approaches is that they are sensitive to asymmetric bandwidths, as the sending rate of payload packets depends on the arrival rate of acknowledgements. We therefore prefer a rate-based approach. Inspired by TCP emulation at receivers [15], we propose a receiver-driven protocol where node B continuously sends a *suggested sending rate* $S_{suggested}^{A \to B}$ to node A as feedback. In this protocol, node B continuously performs the following steps:

1) Observe the flow of incoming packets for a certain observation interval.
2) Update $S_{suggested}^{A \to B}$ based on the measured loss ratio.
3) Wait until the flow of packets coming from A has been adapted.

Whenever node A receives a packet from node B, it simply changes its current sending rate to $\max(S_{goal}^{A \to B}, S_{suggested}^{A \to B})$.

Instead of the rather primitive approach of decreasing the suggested sending rate immediately after a packet loss occurred, we chose a slightly delayed reaction in order to adapt the suggested rate proportional to the current packet loss. This enables our algorithm to adjust the sending rate according to how strong the path is actually congested. Therefore, it avoids an unnecessarily strong lowering of the sending rate, which in turn would have a negative impact on the TCP congestion control of the transported data.

The length of the observation interval acts as a trade-off between responsiveness to packet loss and accuracy of packet loss estimation and therefore needs to be chosen carefully. Consequently, using a fixed period of time to measure packet loss is not useful, as the chosen duration might be too long for one connection and too short for another at the same time. In particular, transmissions over complex network infrastructures, such as the Internet, are likely to experience different latencies depending on the endpoints of communication and the characteristics of the network paths. We chose an automated solution to this problem: node B periodically derives its observation interval from an exponentially weighted moving average of the round-trip time (RTT). In our protocol, node B therefore additionally embeds a *round number* $R_{current}^{A \to B}$ into every packet to node A. Node A in turn simply reports the highest recently received round number $R_{recent}^{A \to B}$ back to node B. To obtain RTT samples, node B always increases $R_{current}^{A \to B}$ by one at the end of the second step and waits until the first packet with the updated round number from A arrives. The arrival of the first packet from A with the updated round number then also marks the end of step 3. However, since A embeds the reported round number in its regular packets, it might not immediately send a packet after it has received the new round number from B (as it depends on its own sending rate). Our measurement procedure therefore slightly overestimates the actual RTT.

The detailed behavior of node B upon arrival of a packet from node A is given in Algorithm 1. The variables listed at the beginning of the pseudo-code represent B's internal state specific to the connection $A \to B$ together with its initial values. To improve readability however, the additional notation of $A \to B$ at each variable was omitted. Furthermore, the minimum observation interval O_{min}, the minimum sending rate S_{min}, the fixed amount S_{step} by which the suggested sending rate is increased and the RTT weight factor w are pre-configured values.

One minor addendum required for Algorithm 1 is that node A also reports its current sending rate $S^{A\to B}_{current}$ to node B. While this data is not a necessity, its availability allows for a much simpler computation of the suggested transmission interval. In our scheme, every packet sent by a node A to its neighbor B carries four values: A's current sending rate $S^{A\to B}_{current}$, A's highest round number $R^{A\to B}_{rcnt}$ recently received from B, the suggested sending rate $S^{B\to A}_{suggested}$ for node B and the current round number $R^{B\to A}_{current}$ for node B. We assume that the underlying VPN mechanism already provides a each sequence number SN_{pkt} for every packet.

The three steps described at the beginning of this section are reflected in Algorithm 1 by the following conditions:

1) While $Updating = false \wedge SN_{pkt} < SN_{thresh}$, node B monitors the incoming packets.
2) When $Updating = false \wedge SN_{pkt} \geq SN_{thresh}$, node B computes the observed packet loss and updates $I_{suggested}$ as well as $R_{current}$ accordingly. To be robust against packet reordering during the observation period, Algorithm 1 only considers the highest recently received sequence number SN_{pkt} and the number SN_{start} that marked the beginning of the current observation interval, together with the total number of received packets at each point (line 12). Since a lower sending rate results in a lower bandwidth consumption, the recently reported value for $S_{current}$ is decreased proportionally to the fraction of lost packets with a cut-off at the predefined minimum sending rate (line 20).
3) While $Updating = true$, node B waits until the first packet with the updated round number from node A arrives. As soon as the round number R_{recent} sent by node A equals the current round $R_{current}$, node B updates the length of the observation interval O (line 3), computes a new value for SN_{thresh} (line 8) and restarts with the first phase.

In contrast to simply relying on a timer to end the observation period, we chose a sequence number threshold SN_{thresh} because of its sensitivity to the current length of the queue at the path bottleneck. If the bottleneck is overloaded and the queue is building up, packets with a sequence number that triggers a sending rate update are delayed, thus reducing the risk of a premature rate increase.

The exact configuration, e.g. maximum queue length and queue management strategy, of the routers on a communication path over the Internet is typically unknown. As a consequence, the delay between the beginning of a network overload and the time when the first packet loss is recognized by the receiver can only be determined heuristically. It is therefore possible that a VPN node running Algorithm 1 underestimates this delay and increases $S_{suggested}$ too early, which leads to the subsequent observation of packet loss before a packet with the updated round number was received. Algorithm 1 therefore uses two additional variables named allow$_{dec}$ and allow$_{inc}$ to allow the receiver to retrospectively correct its feedback $S^{A\to B}_{suggested}$, although with lower accuracy.

Algorithm 1: `ProcessPacket`$(SN_{pkt}, R_{rcnt}, S_{current})$

State:

$O = O_{min}$
$S_{suggested} = S_{init}$
$SN_{thresh} = S_{init} \times {}^{O_{min}}/_{1000}$
$SN_{start} = 0$
$n_{start} = 0$
$n_{recv} = 0$
$t_{last} =$ time of connection establishment
$R_{current} = 0$
Updating $= false$
allow$_{dec} = true$
allow$_{inc} = true$

1 $n_{recv} \leftarrow n_{recv} + 1$
2 $t_{now} \leftarrow$ current system time in milliseconds
3 **if** *Updating and* $R_{rcnt} = R_{current}$ **then**
4 Updating $\leftarrow false$
5 allow$_{dec} \leftarrow true$
6 allow$_{inc} \leftarrow true$
7 $O = \max(O_{min}, O \times w + (1-w)(t_{now} - t_{last}))$
8 $SN_{thresh} \leftarrow SN_{pkt} + O \times S_{current}$
9 $n_{start} \leftarrow n_{recv}$
10 $SN_{start} \leftarrow SN_{pkt}$
11 **if** $SN_{pkt} \geq SN_{thresh}$ **then**
12 $p_{loss} \leftarrow 1 - {}^{n_{recv} - n_{start}}/_{SN_{pkt} - SN_{start}}$
13 **if** $p_{loss} = 0$ *and* allow$_{dec}$ **then**
14 $S_{suggested} \leftarrow S_{current} + S_{step}$
15 allow$_{dec} \leftarrow false$
16 $t_{last} \leftarrow t_{now}$
17 $R_{current} \leftarrow R_{current} + 1$
18 Updating $\leftarrow true$
19 **else if** $p_{loss} > 0$ *and* allow$_{inc}$ **then**
20 $S_{suggested} \leftarrow \max(S_{min}, S_{current} \times (1 - p_{loss}))$
21 allow$_{dec} \leftarrow false$
22 allow$_{inc} \leftarrow false$
23 $t_{last} \leftarrow t_{now}$
24 $R_{current} \leftarrow R_{current} + 1$
25 Updating $\leftarrow true$

V. Evaluation

In this section, we investigate how well our proposed protocol meets the requirements stated in Sec. II. We start with a discussion regarding possible leakage of information by the protocol. In Sec. V-B, we present a measurement study based on a network testbed and discuss its results.

A. Information Leakage

Based on the assumption that VPN systems provide a leakage-free enforcement of a given sending rate, there are two characteristics managed by congestion control for a node A that could leak information to an outside attacker: the difference between the current sending rate $S^{A\to B}_{current}$ before and after a transmission rate update $S'^{A\to B}_{current}$ and the duration between two consecutive transmission rate changes.

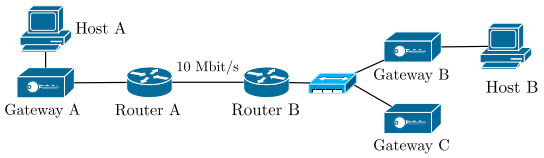

Figure 4. Topology of the testbed used for measurements.

As natural for congestion control, $S'^{A\to B}_{current} - S^{A\to B}_{current}$ might either be positive or negative, depending on the rate suggested by the other endpoint B. In Algorithm 1, each change to the sending rate suggested by B is determined solely based on the packet loss computed from the number of packets received from A and their sequence numbers, as indicated by lines 12-25. The number of received packets is controlled only by the sending rate of node A and the packet loss on the network path, which both cannot be manipulated by a device within the protected network area of node A or B. We assume that the received sequence numbers are assigned by the VPN gateways in a manner that is independent of the characteristics of the encapsulated packets. If B chooses to increase $S^{A\to B}_{suggested}$, A will either set its sending rate to $S^{A\to B}_{suggested}$ or to $S^{A\to B}_{goal}$, as explained in Sec. IV-C. Since $S^{A\to B}_{suggested}$ is computed only from publicly visible information and $S^{A\to B}_{goal}$ is fixed, there is no leakage of information about the internal traffic. Whenever B decreases $S^{A\to B}_{suggested}$, A simply accepts the suggested interval. Thus, an increase in A's sending rate also cannot leak information about the actual internal traffic.

According to lines 7 and 8, Algorithm 1 chooses SN_{thresh} so that each observation period approximately lasts one RTT. The duration between consecutive transmission rate changes can therefore only become shorter if the network's RTT decreases. If the duration between consecutive transmission rate changes gets longer, this could be due to increased network latency, packet loss or both.

Neither of these internal values can be controlled by nodes within the protected network area, thus no information can be leaked to outside attackers. Note that the congestion control algorithm does not take bandwidth consumption of flows by protected nodes into consideration, so that the intensity of data packets being sent by the nodes does not have any effect. When packets are fully encrypted and properly padded, covert channels from the protected network area are ruled out.

B. Utilization of Available Bandwidth

Probably the most fundamental difference of the scenario from previous works is that the congestion control protocol shall not only reach a bandwidth consumption close to the bottleneck bandwidth, but also enable protected TCP flows to attain high goodput. Hence, the measurements presented in the following focus on the interaction between competing VPN flows and the resulting goodput of tunneled TCP transmissions.

To assess our protocol, we conducted experiments in a testbed network consisting of seven nodes as illustrated by Fig. 4. Hosts as well as routers were running a Debian Linux (Kernel `3.16.0-4-amd64`) and `cubic` as the TCP congestion control algorithm (default value). Gateway A, B and C run a micro-kernel-based operating system with a static scheduling plan, so that packets may be sent every 2 ms. The current transmission interval is mapped to a number of packets being sent in each transmission slot. The packets sent by the VPN gateway are UDP-encapsulated ESP packets, padded to a length of 1460 bytes (without the Ethernet header). To create a bottleneck, we reduced the transmission speed of the network interface between Router A and B to 10Mbit/s, but all interfaces were kept in full duplex operation.

To find out how much packet rate enforcement alone already interferes with TCP congestion control, we started with a setup where only gateway A and B are connected, consuming a gross bandwidth of 8.16 Mbit/s in each direction and hence just avoiding any congestion. Goodput was measured by transferring 50 MB of data from Host A to Host B using `iperf` (version 2.0.5). Averaging over 32 runs, the achieved goodput amounts to 7.35 Mbit/s with a standard deviation of 0.01 Mbit/s. Due to the overhead created by UDP encapsulation, ESP and congestion control data, each protected packet is only 1370 bytes long, so that effectively 1318 bytes of payload can be transferred per packet. The maximum goodput that can be achieved with the fixed sending rate is therefore 7.36 Mbit/s. Consequently, the rate enforcement alone had a negligible impact on TCP goodput according to the measurements.

For the evaluation of the interaction between our protocol and the protected TCP traffic during congestion, we consider the two cases that congestion occurs only in one direction of the communication path, and that both directions are congested. In the unidirectional congestion scenario, we set $S^{A\to B}_{goal}$ and $S^{A\to C}_{goal}$ to 8.16 Mbit/s (gross) and $S^{B\to A}_{goal}$ as well as $S^{C\to A}_{goal}$ to 4.08 Mbit/s so that both flows originating at node A compete for bandwidth. For the bidirectional scenario, we again set $S^{B\to A}_{goal}$ and $S^{C\to A}_{goal}$ to 8.16 Mbit/s. During all measurements, the values S_{min} and S_{step} were set to 204 kbit/s. Furthermore, O_{min} and w were set to 40ms and $1/8$, respectively. For the value of p_{loss}, we used a fixed-point number ranging from 0 to 10.000. We used the default drop-tail as queue management algorithm for the interface connecting Router B to Router A and vice versa. The limit for the length of each queue was set to 20 packets.

Table I
PERCENTAGE OF RETRANSMISSIONS AND AVERAGE RTT UNDER CONGESTION DEPENDING ON CONGESTION CONTROL.

Cong. control	Cong. type	% Retransm.	RTT
disabled	unidirectional	$24.4 \pm 1.2\%$	44.5 ± 0.43 ms
enabled	unidirectional	$1.9 \pm 0.05\%$	42.49 ± 0.26 ms
	bidirectional	$1.87 \pm 0.14\%$	70.75 ± 0.89 ms

Again, we measure the achieved goodput by transferring a fixed amount of bytes from Host A to Host B using `iperf`. Fig. 5 summarizes the achieved goodput of each algorithm for the different congestion scenarios. The height of each bar denotes the average goodput and the lines around the top of each bar represent the observed minimum and maximum. For

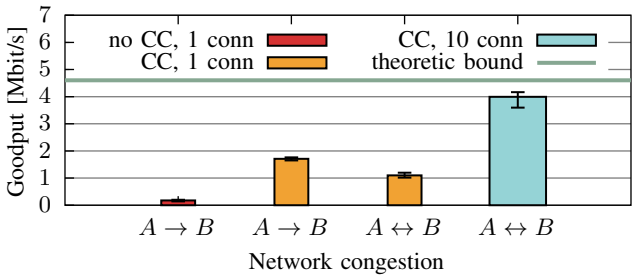

Figure 5. Measured TCP goodput for different congestion setups and congestion control schemes, showing: the situation without our system (red), performance of a signle flow under uni- and bidirection congestion (orange) and the accumulated goodput of 10 flows (turquoise). Note the theoretic upper bound of 4.6 Mbit/s.

comparison, there is a horizontal line at 4.60 Mbit/s, as this would be the goodput achieved in case of ideal congestion control. In case of unidirectional congestion and no congestion control, TCP only achieved an average goodput of 175 ± 0.013 kbit/s due to high packet loss. Results for bidirectional congestion were omitted, as even the time required for a transmission of 10 MB exceeded the time available for the measurement study. After integration of the Algorithm 1, the average TCP goodput increased to 1.7 ± 0.04 Mbit/s under unidirectional congestion. Under bidirectional congestion, the average TCP goodput increased to 1.1 ± 0.06 Mbit/s. To clarify the reasons for these results, Table I shows the average percentage of retransmissions and the average round-trip time experienced during the TCP transmissions. A rather prominent result is that the congestion control only slightly influences the RTT, but may heavily reduce the number of retransmits.

The results presented so far showed what a single TCP connection can achieve under our congestion control scheme. However, it seems likely that most of the time, there are multiple competing TCP transmissions running over the VPN connection. Therefore, we also measured the goodput of 10 concurrently running TCP transmissions from Host A to Host B under bidirectional congestion.

Summarizing these measurements, the flow of packets from gateway A to gateway B allowed a minimal, mean and maximal goodput of 4.30 Mbit/s, 4.49 Mbit/s and 4.68 Mbit/s, respectively. The allowed goodput G_{max} is derived from the consumed bandwidth of the traffic $T_{consumed}$ flowing from gateway A to gateway B and the overhead incurred by ESP and TCP according to the formula $G_{max} = T_{consumed} \times {}^{1318}/_{1474}$. From the allowed goodput, the TCP concurrent transmissions were able to cumulatively utilize at least 3.59 Mbit/s, 3.99 Mbit/s on average and at most 4.16 Mbit/s. Consequently, it can be said that even when the packet loss resulting from congestion control by the VPN gateways lowers the goodput of protected TCP flows, a high percentage of the consumed bandwidth can still be utilized if there are multiple simultaneous transmissions.

VI. Conclusion & Future Work

In this paper, we proposed a congestion control scheme that extends strict traffic normalization by achieving automatic adaptation to the current network transport capacity without leaking information about the protected traffic. The control scheme first obtains an estimate of the current packet loss rate and then adjusts the sending rate proportionally, hence avoiding strong rate throttling in response to packet loss. To adapt the packet loss estimation to the network's latency characteristics, the scheme waits until a dynamically chosen number of packets have been received. Leakage of information is completely avoided, since the control algorithms work solely based on publicly visible information. We furthermore conducted a measurement study which confirmed that when congestion occurs, applying our proposed control scheme indeed yields a significantly increased goodput of TCP connections compared to traffic normalization without congestion control. Thus, the scheme may help for traffic normalization to become more acceptable.

The measurement study also revealed that our algorithm currently tends to recognize congestion slightly too late. One part of our future work will therefore be to improve how the number of packets needed for loss estimation is chosen. We furthermore plan to integrate Explicit Congestion Notification into the control scheme to reduce the experienced packet loss and therefore improve the goodput of protected TCP flows.

Acknowledgements

This work was in parts supported by BMBF project AN.ON-next under Grant No. 16KIS0421.

References

[1] G. Greenwald, "The U.S. Government's Secret Plans to Spy for American Corporations," https://theintercept.com/2014/09/05/us-governments-plans-use-economic-espionage-benefit-american-corporations/, 2014.
[2] M. Backes, G. Doychev, M. Dürmuth, and B. Köpf, "Speaker Recognition in Encrypted Voice Streams," in *ESORICS*, 2010.
[3] G. Shah, A. Molina, M. Blaze *et al.*, "Keyboards and Covert Channels," in *Usenix Security*, 2006.
[4] S. Wendzel, S. Zander, B. Fechner, and C. Herdin, "Pattern-Based Survey and Categorization of Network Covert Channel Techniques," *ACM Computing Surveys*, vol. vol. 47, issue 3, 2015.
[5] F. Rezaei, M. Hempel, P. Shrestha, S. Rakshit, and H. Sharif, "Detecting Covert Timing Channels using Non-Parametric Statistical Approaches," in *IEEE IWCMC*, 2015.
[6] S. Cabuk, C. E. Brodley, and C. Shields, "IP Covert Timing Channels: Design and Detection," in *Proceedings of the 11th ACM conference on Computer and Communications Security*, 2004.
[7] H. Zhao and Y.-Q. Shi, "Detecting Covert Channels in Computer Networks based on Chaos Theory," *IEEE TIFS*, vol. vol. 8, no. 2, 2013.
[8] S. Gianvecchio and H. Wang, "Detecting Covert Timing Channels: An Entropy-Based Approach," in *ACM CCS*, 2007.
[9] R. Browne, "Mode security: An Infrastructure for Covert Channel Suppression," in *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, 1994.
[10] Y. Guan, X. Fu, D. Xuan, P. U. Shenoy, R. Bettati, and W. Zhao, "NetCamo: Camouflaging Network Traffic for QoS-Guaranteed Mission Critical Applications," *IEEE TSMC*, vol. 31, 2001.
[11] M. H. Kang, I. S. Moskowitz, and D. C. Lee, "A network pump," *IEEE Transactions on Software Engineering*, vol. vol. 22, issue 5, 1996.
[12] A. Sadeghi, S. Schulz, and V. Varadharajan, "The Silence of the LANs: Efficient Leakage Resilience for IPsec VPNs," in *ESORICS*, 2012.
[13] J. Widmer, R. Denda, and M. Mauve, "A Survey on TCP-Friendly Congestion Control," *IEEE Network*, vol. vol. 15, issue 3, 2001.
[14] S. Floyd and K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet," *IEEE/ACM Transactions on Networking*, vol. vol. 7, issue 4, 1999.
[15] I. Rhee, V. Ozdemir, and Y. Yi, "TEAR: TCP Emulation at Receivers–Flow Control for Multimedia Streaming," Tech. Rep., 2000.