INTERNATIONAL JOURNAL OF NETWORK MANAGEMENT
Int. J. Network Mgmt 2013; **23**: 16–40
Published online 1 April 2012 in Wiley Online Library (wileyonlinelibrary.com) DOI: 10.1002/nem.1806

A cognitive accountability mechanism for penalizing misbehaving ECN-based TCP stacks

Steven Latré,[1,*†] Wim Van de Meerssche,[1] Dirk Deschrijver,[1] Dimitri Papadimitriou,[2] Tom Dhaene[1] and Filip De Turck[1]

[1]*Department of Information Technology, Ghent University - IBBT, 9050 Ghent, Belgium*
[2]*Alcatel-Lucent Bell Labs, B-2018 Antwerp, Belgium*

SUMMARY

The introduction of high-bandwidth demanding services such as multimedia services has resulted in important changes on how services in the Internet are accessed and what quality-of-experience requirements (i.e. limited amount of packet loss, fairness between connections) are expected to ensure a smooth service delivery. In the current congestion control mechanisms, misbehaving Transmission Control Protocol (TCP) stacks can easily achieve an unfair advantage over the other connections by not responding to Explicit Congestion Notification (ECN) warnings, sent by the active queue management (AQM) system when congestion in the network is imminent. In this article, we present an accountability mechanism that holds connections accountable for their actions through the detection and penalization of misbehaving TCP stacks with the goal of restoring the fairness in the network. The mechanism is specifically targeted at deployment in multimedia access networks as these environments are most prone to fairness issues due to misbehaving TCP stacks (i.e. long-lived connections and a moderate connection pool size). We argue that a cognitive approach is best suited to cope with the dynamicity of the environment and therefore present a cognitive detection algorithm that combines machine learning algorithms to classify connections into well-behaving and misbehaving profiles. This is in turn used by a differentiated AQM mechanism that provides a different treatment for the well-behaving and misbehaving profiles. The performance of the cognitive accountability mechanism has been characterized both in terms of the accuracy of the cognitive detection algorithm and the overall impact of the mechanism on network fairness.

Received 15 February 2011; Revised 10 February 2012; Accepted 16 February 2012

1. INTRODUCTION

Today's Internet has undergone a major shift since its original adoption: narrowband services such as web browsing and e-mails are no longer the most popular services. Instead, they have been replaced with multimedia services such as video streaming and online gaming, which are all very bandwidth intensive and require strict quality-of-service (QoS) guarantees to ensure a smooth service delivery. Even small drops in bandwidth can deteriorate the quality as perceived by the end user, often called the quality of experience (QoE). For example, a drop in bandwidth of a video transmission can lead to visual artifacts or play-out buffer starvation that are visible through freezes of the video.

As the demand for bandwidth in the Internet grows, and especially in access networks, so does the number of connections with a large volume [1]. Although short-lived connections (mice) are still plentiful, the introduction of video services also results in an increasing number of long-lived connections (elephants). For example, the popular HTTP Adaptive Streaming techniques use HTTP connections that last at least a

*Correspondence to: Steven Latré, Department of Information Technology, Ghent University - IBBT, Gaston Crommenlaan 8/201, 9050 Ghent, Belgium.
[†]E-mail: Steven.Latre@intec.ugent.be

few seconds to transmit segments of the videos. One connection is often reused for the transmission of multiple video segments, which leads to connections that can last for minutes and transmit megabytes of data. These longer-lasting connections increase the need for techniques that ensure an adequate fairness between connections. The longer connections last, the more they can take advantage of their achieved rate (e.g. by starving newly started connections) as they are further away from their start-up phase, which is often characterized by a slow start behavior. Enforcing fairness between connections is therefore becoming increasingly important in today's Internet.

Today, TCP is by far the most widely used transport protocol, especially for video services, where increasingly progressive download-based mechanisms are used. Although the widely used TCP protocols all contain congestion avoidance algorithms that, in theory, should enforce fairness between connections, practice has shown that existing differences between TCP stacks result in important limitations. This is motivated by one of the Internet Research Task Force's recent Requests for Comments (RFCs) [2], discussing the open research issues in Internet congestion control, which identifies misbehaving senders and receivers as one of the seven main challenges in congestion control. Furthermore, Sherwood *et al*. [3] show that the presence of a single misbehaving connection can lead to a collapse of the throughput of other connections. According to them, in the worst case, the achieved gain of the misbehaving connection is a factor of 30 million. As TCP only resides on the terminals of a connection, a network relies on the correctness of the TCP's congestion avoidance algorithms. However, it is not in the TCP stack's best interest to feature such altruistic behavior. Therefore, TCP implementations can exploit this trust to achieve a higher throughput while violating TCP's congestion avoidance principles, at the cost of the throughput of other connections. As we will show in this article, such behavior can even be inflicted without modifying the implementation of a TCP/IP stack in the kernel. TCP stacks that feature these limitations are called misbehaving, because they do not comply with the agreed upon reaction to congestion signals. Without in-network management, it is not possible to adequately manage misbehaving TCP stacks.

Instead of solely trusting to the congestion avoidance behavior of the terminals, more intelligence is often introduced into the network's routers through the active queue management (AQM) paradigm. Instead of only dropping packets when a router's queue is full, an AQM system can drop packets earlier or can signal the existence of congestion to the terminals through the Explicit Congestion Notification (ECN) [4], which marks the packets by setting a flag in the packet header. The TCP sender is then expected to respond to this congestion signal by lowering its rate, provided that the TCP receiver behaves properly and echoes the congestion signals. Additionally, by assigning the connections to different traffic classes, a router can also choose to prioritize one traffic class over another.

The flexibility provided by an AQM system can thus be exploited to introduce a greater level of fairness between connections. If it is possible to detect misbehaving TCP stacks, an autonomic differentiated AQM management can be introduced that distinguishes between well-behaving and misbehaving stacks to penalize misbehaving TCP connections and reward well-behaving TCP connections. As this detection consists of finding different behavior between connections, we argue that a cognitive approach is best suited. In this article, we provide an answer to the following research questions: (i) How can the occurrence of misbehaving TCP stacks be detected inside a router's AQM system? (2) Can the penalization of misbehaving connections lead to a better fairness between connections? (3) How fast and accurate is the response of a differentiated AQM management mechanism?

In this article, we propose a cognitive mechanism, which we call a cognitive accountability mechanism, that is able to detect misbehaving TCP senders and receivers and penalizes them to achieve a better fairness between the individual connections. The cognitive mechanism can be deployed on top of a traditional AQM management scheme and is transparent to the other nodes in the network. It is mainly targeted for deployment in an access network environment, where the gain that can be achieved by misbehaving is considerable. As such, it specifically targets environments where the connections have a considerable duration (i.e. more than 1 second) and a moderate number of concurrent connections (i.e. a few thousand or less). The cognitive accountability mechanism provides a service to the operator by altering the forwarding of packets to improve fairness. The main idea is that the algorithm holds subscribers accountable for their actions by penalizing misbehaving connections. A general overview of the proposed cognitive mechanism is illustrated in Figure 1. The major contributions of this article are the following. First, we propose a machine-learning based

DOI: 10.1002/nem

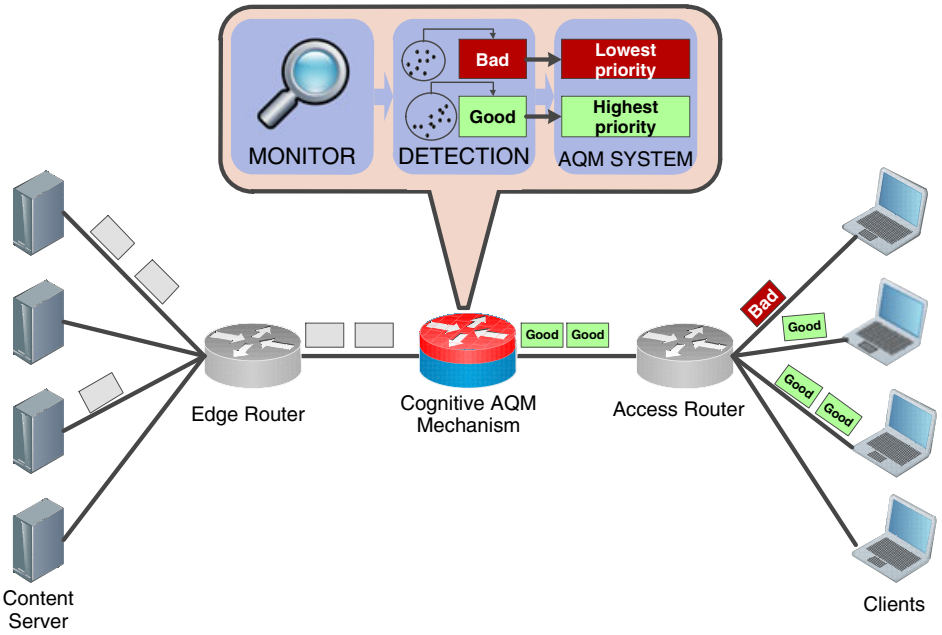

Figure 1. Overview of the role of a cognitive mechanism on top of an existing AQM system to introduce a higher level of fairness. Individual TCP connections are monitored to detect misbehaving TCP stacks (senders or receivers), which can thenbe penalized to favor the well-behaving TCP stacks

detection algorithm that monitors the traffic passing through a router and decides whether or not a TCP stack is misbehaving. The detection algorithm uses a combination of outlier detection and clustering based on extracted flow statistics to find different behavioral profiles and an intelligent labeling mechanism to distinguish between well-behaving and misbehaving stacks. Second, we present a differentiated AQM mechanism that penalizes misbehaving stacks by treating the misbehaving flows as non-ECN flows. Third, we evaluated the cognitive mechanism by deploying it on an NS-2-based simulation environment under different network configurations (e.g. multiple round trip time (RTT) configurations). The performance evaluation characterizes the accuracy of the detection algorithm and identifies the obtained gain of the differentiated AQM mechanism in terms of improved fairness for different types of misbehaving stacks.

The remainder of this article is structured as follows. Section 2 discusses relevant work in the fields of congestion control and machine-learning based management. Section 3 characterizes the destructive effect of misbehaving TCP stacks on the obtained fairness. Sections 4 and 5 present the cognitive mechanism's detection algorithm and differentiated AQM mechanism, respectively. Both the detection and penalization algorithms are evaluated in Section 6 and the article is concluded in 7.

2. RELATED WORK

Congestion control algorithms are probably the most widely available feedback loop algorithms available in the current Internet. As the Internet connects users from all over the world, competing for the same bandwidth, already from the Internet's early adoption it became clear that, in order to guarantee a reliable and fair data transfer, the rate at which the competing sources were sending needed to be controlled. Owing to the distributed nature of the Internet and thus the lack of a centralized manager, these congestion control algorithms must apply adaptive heuristic algorithms that estimate the network load and react by changing their rate on the sign of congestion. Some key papers from Jacobson [5] and Chiu and Jain [6] highlighted the gain of an adaptive algorithm, called Additive Increase Multiplicative Decrease (AIMD), which linearly increases the rate in the absence of congestion and exponentially decreases the rate in the presence of congestion.

DOI: 10.1002/nem

In today's Internet, congestion control algorithms are typically deployed both on the terminals (i.e. the end users) of the network as in the network itself. At the terminals, congestion control in the TCP protocol was standardized in Allman *et al*. [7] and the concept of a congestion window, denoting the maximum rate at which a sender may transmit, was introduced. Since then, congestion control algorithms have found their way into all TCP implementations used, such as TCP New Reno [8] and TCP CUBIC.

In the network itself, the introduction of AQM has led to a finer control on the traffic that passes through the routers. Instead of using DropTail queues, which simply drop packets once the queue is full, random early detection (RED) [9] algorithms and variants such as BLUE [10] maintain an exponentially weighted average value of the queue length and base their dropping probability on the average queue length. These alternate queuing algorithms have the effect that the amount of packet drops, which is a signal of congestion for most TCP congestion control algorithms, exponentially increases as the load increases. By combining RED with the ECN mechanism, congestion warnings other than packet drops can be given to TCP terminals without inflicting data loss.

The growing diversity in congestion control on one hand and the increasing importance of achieving fairness on the other has led to modeling approaches where the interaction between TCP congestion control and AQM is analytically modeled [11,12]. Kelly *et al*. [13] showed that a TCP congestion control algorithm on one hand and an AQM scheme on the other can be modeled as two separate components—a primal and dual component—that both try to maximize a utility function. Depending on the implementation of the congestion control algorithm, other utility functions can be provided. In striving to maximize their separate utility functions, Kelly showed that they reach an equilibrium which corresponds to different fairness levels depending on the specifics of the utility functions.

While these different levels of fairness can be characterized offline with knowledge of the implementation details of the congestion control algorithms, for a router in an online scenario there is no way of knowing these utility functions and the corresponding achieved fairness. Therefore, this is a suboptimal solution for a network provider: he typically wants network fairness that does not depend on the type of congestion control algorithms used. As he has no control over the terminals in the network, misbehaving TCP stacks can deteriorate the quality of other, well-behaving, TCP stacks [14].

In this article, we focus on unresponsive flows caused by misbehaving ECN senders or receivers. This type of misbehaving has received much attention as the ECN mechanism can easily be exploited to ignore ECN signals without requiring any change in the kernel's implementation. This has led to an extension of ECN, called ECN-nonce, defined by the IETF in Spring *et al*. [15]. However, to the best of the authors' knowledge, ECN-nonce has not yet been significantly deployed. ECN-nonce enables (a sender) to check the integrity of a receiver. Kulatunga and Fairhurst [16] extend the ECN-nonce technique to support also misbehaving receivers in multicast scenarios. However, ECN-nonce relies on the integrity of the server, which is sometimes not the case. Our approach assumes both misbehaving senders and receivers. Moreover, performing the same integrity check by an AQM system in a router can be challenging. Our approach is specifically designed to be deployed on top of an existing AQM system. As such, we believe that our approach can be complementary to ECN-nonce, where ECN-nonce is deployed at the end systems and our approach provides an additional check (also including misbehaving senders) in the network.

More recently, the need for effective congestion management by Internet service providers (ISPs) and a possible way forward has been published in an informational RFC (RFC6057 [17]). In RFC6057, the connection management system of a large cable broadband ISP, Comcast, is discussed. While the approach is specifically applied to cable networks, its approach contains enough generic blocks to be applicable to other networks as well. The Comcast approach tries to improve the network fairness by lowering the priority of users that caused high volumes of traffic during recent minutes. The effect is that these users are more likely to observe packet loss when congestion occurs. Similar to the Comcast approach, our approach also penalizes users that cause high volumes of traffic by changing their priority. In our case, disabling their ECN marking enforces this. However, our approach differs from the Comcast approach in two ways. First, we focus on misbehaving senders and receivers that deliberately want to increase the traffic at the cost of others. Instead, the Comcast also penalizes users who diverge from the average amount of throughput that can be observed in the system. Second, we base our decision not only on whether or not to penalize users based on the achieved throughput.

DOI: 10.1002/nem

Instead, other flow statistics such as the RTT or ECN statistics are taken into account as well. This makes our approach less protocol agnostic, but increases the accuracy and effectiveness of the approach considerably.

Recently, several extensions have been proposed that optimize TCP congestion avoidance algorithms for ECN-enabled connections. These extensions alter the way end-terminals respond to the ECN congestion signals. The ECN-hat proposal [18] allows a TCP sender to cut its congestion window by a smaller factor than 2 upon reception of a congestion signal, as echoed by the TCP receiver. ECN-hat maintains an exponentially weighted moving average that denotes the reduction factor, which is between 1 and 2. An evolution of the ECN-hat proposal, called Data Center TCP [19], allows a TCP receiver not only to echo the *existence* of congestion but also the *extent* of that congestion. The TCP sender uses this echoed information to enable a more fine-grained control of the congestion window. As both of these proposals are (i) still in an experimental phase and (ii) have a deployment in data centers as main target (in contrast to our access network deployment target), they were not studied in this article. However, as they also influence the way ECN signals are echoed and how TCP senders respond to these ECN signals, they can potentially result in important detection differences. A performance evaluation of this type of extensions is therefore an interesting part of future work.

One interesting approach that has recently been proposed to increase fairness in a connectionless network is the introduction of more detailed causal congestion information. The Congestion Exposure (ConEx) approach [20,21], originally introduced by Bob Briscoe [22], enables TCP end-terminals to truthfully expose the congestion level they are expected to introduce into the network. The ConEx protocol can be seen as an independent protocol, which is orthogonal to techniques such as ECN. The Re-ECN protocol [23] is an implementation of the ConEx concept in an ECN-enabled environment. Re-ECN reinserts the congestion feedback that was received from the TCP receivers into the forwarding path. Based on this more detailed feedback information, AQM algorithms can provide more intelligent and effective actions to maximize the fairness between connections. As such, re-ECN is in itself not a solution to fairness but provides the first step towards more intelligent actions. In terms of deployment, re-ECN requires modifications to the sender itself. Alternatively, a proxy can mimic the required modifications to the sender if needed. For more information about re-ECN, we refer to Briscoe and co-workers [24,25]. With respect to our approach, the inclusion of added feedback into the downstream direction can only increase the accuracy of detecting misbehaving TCP senders or receivers. However, changes would be necessary to the detection algorithm to incorporate this additional feedback. As such, both techniques should positively influence each other. In this article, we focus on the currently deployed combination of ECN and TCP congestion avoidance techniques: the combined study with Re-ECN or other, alternative, ConEx implementations is part of future work.

Huang *et al*. [26] and Hanlin *et al*. [27] address the issue of improving the fairness between connections, either through the design of a new AQM algorithm or the adaptation of RED, respectively. Both approaches are focused on penalizing misbehaving connections in the form of misbehaving UDP connections. The Pre-Congestion Notification mechanism (PCN) [28], which is a measurement-based admission control system, also supports flow termination. PCN is mainly targeted at inelastic (e.g. UDP-based) flows. Thus PCN can perform penalization actions to unresponsive UDP flows. In this article, we focus on misbehaving TCP connections. Kuzmanovic *et al*. [29] study misbehaving TCP receivers. However, their work focuses on TCP congestion control algorithms that are receiver oriented. They show that receiver-driven TCP has its potential but that a balance must be found between enforcement mechanisms and a complete trust of the endpoints. The issue of misbehaving behavior in the generation of SACKs is studied by Ekiz *et al*. [30]. The approach uses TBIT to fingerprint connections that misbehave in the SACK generation. Their main goal is to identify and document the types of misbehaving stacks, not to correct them. In Cheng and Lin [31], misbehaving TCP receivers are studied. The authors present a modification to the original TCP protocol to deal with different types of misbehaving receivers. Similarly, Chan *et al*. [32] address misbehaving TCP receivers by generating a secret key at the sender. The TCP sender is thus responsible for enforcing the correctness of the receiver. In contrast to both previous approaches, our approach focuses both on misbehaving senders and receivers. Moreover, our approach is deployed in the network and thus does not require any change to the TCP protocol itself. This facilitates its deployment and incremental adoption.

DOI: 10.1002/nem

We argue that a cognitive approach, consisting of an unsupervised machine-learning algorithm, is most suited to detect misbehaving TCP stacks. This is because of its adaptive nature to changing circumstances and because the detection of a misbehaving stack is in essence a clustering problem that finds the cluster of good stacks versus the cluster of bad stacks. Machine-learning algorithms have been successfully applied to network management domains such as improved routing [33], topology control [34] and anomaly detection [35,36].

More specifically related to the problem of congestion control, Raineri and Verticale [37] use a classification method to identify the type of application. This classification is based on statistical features of the connections such as length of the packets and inter-arrival times. In our approach, we use similar statistical information of a connection to decide on the misbehaving of a connection. In El Khayat *et al*. [38], a classification mechanism is proposed that is able to distinguish between losses related to congestion and losses related to link errors. As many congestion control algorithms interpret packet loss as a sign of congestion, the proposed classification method is able to filter out the data losses that have nothing to do with congestion, which increases the throughput of the TCP connections. Jayaraj *et al*. [39] tackle the same problem but use hidden Markov models and expectation maximization algorithms as machine-learning techniques. Our proposed technique is complementary to these techniques as it can only benefit from detecting misbehaving connections and hence take more accurate decisions on congestion signals.

3. PROBLEM STATEMENT

In this section, we discuss the impact of introducing misbehaving ECN-based senders and receivers into a set of TCP connections. We first provide a summary of the ECN extension and then describe how TCP stacks can misbehave by not responding to ECN signals. Finally, we discuss the experimental setup used and characterize the gain they can achieve by misbehaving.

3.1. Overview of the explicit congestion notification extension

ECN [4] is an extension to the standard IP and TCP protocol. Its main goal is to introduce the ability of explicit signaling to the end terminals when congestion occurs in the network. Without ECN, congestion is signaled implicitly by the dropping of packets. While traditional congestion avoidance algorithms are able to respond to these implicit congestion signals, this often results in unnecessary retransmissions. In turn, these retransmissions lead to an increased latency for those retransmissions, and thus a higher jitter for the affected connections. ECN has the advantage that it alleviates those problems by piggybacking a congestion signal with regular packets instead of dropping those packets.

The congestion signals are sent by marking the regular stream of packets passing through a router. In this context, marking means setting one or more bits in the IP or TCP header. In summary, ECN operates as follows: when congestion occurs in the network, packets are marked by setting the 'Congestion Experienced' flag in the IP header. This flag is raised by setting 2 bits of the DiffServ field in the IPv4 or IPv6 header. Which packets are marked depends on the active queue management algorithm used in the router. For example, in RED, the probability that is calculated to define the dropping probability (if the average queue length is between min and max) also defines the marking probability. Hence the packets are in this case marked instead of dropped. Only under very high load conditions, i.e. if the max threshold of the RED algorithm is exceeded, are packets dropped. While the congestion signal itself (the 'Congestion Experienced' (CE) flag) is encoded in the IP header, the higher layer transport protocol is supposed to respond to these congestion signals. TCP supports ECN through two flags in the TCP header. For ECN-enabled TCP connections, the TCP receiver is expected to echo these congestion signals back to the sender. This is done through the ECN Echo (ECE) flag. The TCP sender is expected to respond to this ECE signal, as for packet drop, by lowering its congestion window and confirming the decrease of the congestion window through a Congestion Window Reduced (CWR) flag.

DOI: 10.1002/nem

3.2. Misbehaving TCP stacks

We introduced different types of misbehaving TCP stacks by altering the response of these TCP stacks to ECN signals. We investigate both misbehaving TCP receivers and TCP senders.

3.2.1. Misbehaving TCP receivers

Partially ignoring ECN messages. A TCP receiver can easily exploit the ECN mechanism by ignoring some or all ECN signals he receives. In this case, the congestion signals are not echoed back to the TCP sender, and he cannot adapt its congestion window, leading to a higher throughput until the load becomes too high and the TCP sender responds to actual packet loss. After that, the rate is again gradually increased until packet loss is again experienced. When only a fraction of the TCP stacks ignore these messages can they achieve a higher throughput, because other TCP stacks will lower their send rate earlier. In this case, the misbehaving TCP stack is positioned at the receiver side instead of the sender side. Note that an implementation of such a misbehaving TCP stack is straightforward and does not even require modifications of the TCP/IP stack in the kernel. Received packets can easily be captured and modified to remove the ECN flag, at which point they are sent to the actual TCP stack. In our experiments, we varied the percentage of ECN messages that are ignored to introduce levels of misbehaving. In the remainder of this article, this type of misbehaving TCP stack is abbreviated as IgnoreRecX, where X is the percentage of ECN messages that are ignored. Hence, IgnoreRec30 will ignore 30% of all ECN messages.

Adaptively ignoring ECN messages. Instead of ignoring random ECN messages, a second type of investigated misbehaving TCP stack performs a more adaptive ignoring of ECN messages. This type of ECN flag manipulation takes place on the receiver side, but assumes that the receiver knows the configuration parameters and current queue size of the router sending the ECN messages at all times. Note that this type of information is hard to estimate for the receiver. However, this type of misbehaving denotes a more advanced ignoring as it continuously switches between a misbehaving and well-behaving mode, which is of interest from a theoretical viewpoint, as it is harder to detect this behavior. In our experiments, the AdaptiveIgnore component was deployed just after the queue to allow it to have access to the queue length.

In this case, the ECN Congestion Experienced flag is only ignored in situations where the queue is not in an extreme load mode. These situations are denoted by the length of the queue with respect to the RED thresholds. The idea is that the misbehaving stack pushes the queue to the limit, without actually causing losses. Therefore, ECN messages are ignored until the queue length is higher than the RED max threshold, at which point a fraction of the packets of the packets are being dropped by the queue. When this RED max threshold is exceeded, the ignoring of congestion warnings is temporarily suspended, which makes the TCP stack behave as a regular TCP stack again. Once the decrease in the queue length is sufficient, denoted by the RED min threshold, the misbehaving is again enabled and all ECN messages are ignored. In the remainder of this article, this type of misbehaving TCP stack is abbreviated as AdaptiveIgnore.

3.2.2. Misbehaving TCP senders

Similar to the misbehaving of a TCP receiver, it is also possible for a TCP sender to misbehave. In terms of ECN behavior, a TCP sender can choose to ignore the messages containing ECE flags and not to respond to these congestion warnings. The rationale here is that the misbehaving TCP sender tries to receive an advantage over the well-behaving TCP senders by depending on the other well-behaving TCP senders to lower their rate. If this succeeds, a misbehaving TCP sender can potentially obtain a higher throughput compared to the other connections. Obviously, even a misbehaving TCP sender will ultimately need to lower its rate. This happens when it increases its rate up to the level that congestion related losses occur. Similarly to the implementation of a misbehaving TCP receiver, this type of misbehaving can easily be implemented without requiring any change to the original TCP/IP stack. It is sufficient to capture packets sent to the TCP sender and unset the ECE flag. In our experiments, we define this type of misbehaving TCP stack as an IgnoreSenX, where X is the percentage of ECE flags that are ignored.

3.3. Experimental setup

Before characterizing the gain misbehaving TCP senders and receivers can achieve by ignoring congestion warnings, we first discuss the experimental environment that will be used throughout this

DOI: 10.1002/nem

article. The above discussed types of misbehaving TCP senders and TCP receivers were implemented in the NS-2 network simulator (version 2.35) [40]. As TCP implementation, we used NS-2's Full TCP implementation with ECN support. The investigated topology is illustrated in Figure 2. The topology models TCP receivers in three domains which are connected to a single domain of TCP senders. There is a bottleneck of 100 Mbps between the TCP sender and the three sites of TCP receivers. Each of the three domains has a configurable delay, denoted RTT_1, RTT_2 and RTT_3. At each site, 25% of all connections are limited by another bottleneck of 20 Mbps.

The start times of the connections were modeled using a Weibull process with scale λ and shape k, which defined the inter-arrival times of the connections in milliseconds. To vary the connection's duration, a Pareto distribution with scale x_m and shape α was used that defines the number of bytes that each connection sends. During the experiments, the parameters were set as follows: $\lambda = 143$, $k = 0.9$, $x_m = 1500$ and $\alpha = 1$. Note that this corresponds to an average inter-arrival time of 0.15 seconds and an average connection duration in the order of a few seconds. The Weibull distribution was modeled based on findings in Pustisek *et al*. [41]. The share of misbehaving nodes (senders or receivers) was varied in the experiments. The assignment of the TCP receivers on the different sites and rate-limiting bottlenecks and whether or not they were misbehaving was done through a uniform distribution. All experiments were repeated 100 times.

3.4. Experienced fairness

We investigated the impact of the three different misbehaving TCP stacks on the experienced fairness. We focus on the gain in throughput they achieve by misbehaving, compared to the well-behaving TCP stacks. We characterize the relative gain: hence a gain of 200% means that the throughput is two times higher than its well-behaving counterparts. In this case, the RTTs of the three sites was set to the same value, i.e. 20 ms.

Figure 3 illustrates the relative gain of misbehaving TCP connections given an increasing number of concurrent connections. In this experiment, we introduced 50% of misbehaving connections into the total connection pool. As shown, the biggest gain can be achieved when the connection pool is limited: with only two connections available, the gain a misbehaving stack achieves can be up to 15 times larger than the concurrent well-behaving stack. As the connection pool increases, the gain that can be achieved by misbehaving decreases. However, even for a moderate connection pool the gain is still notable. For example, for 500 simultaneous connections, a misbehaving TCP sender that ignores all congestion warnings can still increase its throughput by a factor of 2.44 compared to the well-behaving connections. When comparing the different types of misbehaving TCP stacks, we see that TCP receivers achieve a smaller gain than misbehaving TCP senders. This is because the TCP sender

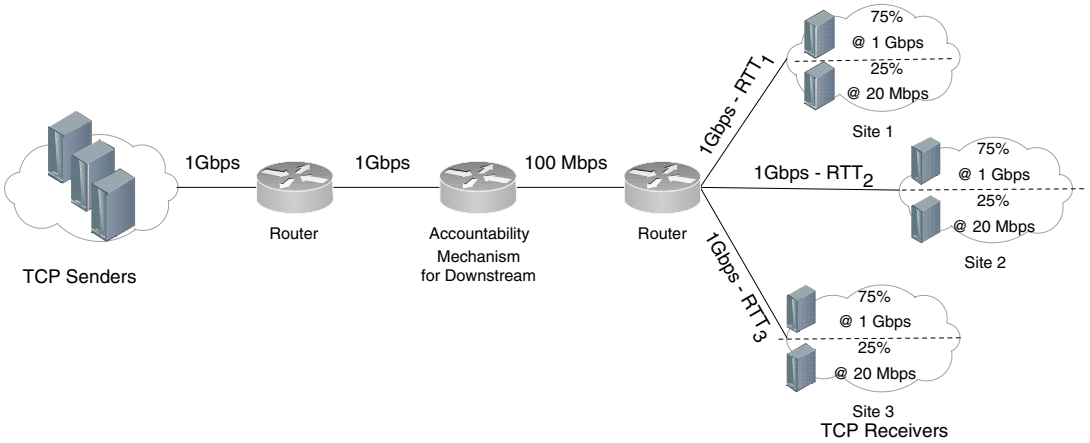

Figure 2. Network topology used for all experiments, with a central bottleneck of 100 Mbps. The topology consists of three sites, with various RTTs, that request content from a single TCP sender site. 25% of all TCP receivers experience an additional bottleneck of 20 Mbps

DOI: 10.1002/nem

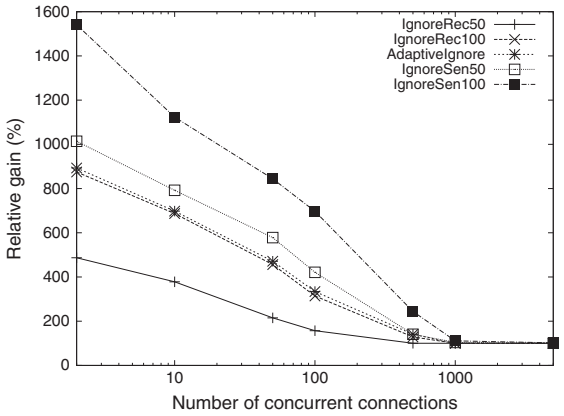

Figure 3. Relative gain of misbehaving TCP stacks given an increasing number of concurrent connections. As the number of consecutive connections is increased, the gain obtained by misbehaving TCP stacks diminishes

controls the rate: if it decides to ignore congestion warnings, the impact on the rate is immediately visible. Additionally, the AdaptiveIgnore type of misbehaving receiver is able to achieve a slightly higher rate than the IgnoreRec100 variant, but the difference is limited.

In the network model used, the number of bytes sent by the TCP connection was varied using a Pareto distribution. Together with a different number of concurrent connections this resulted in a varying connection duration. Figure 4 shows the relative gain of misbehaving stacks as a function of an increasing connection duration of those misbehaving TCP stacks. In this case, the relative gain was averaged over all kinds of connection pool sizes. As shown, short misbehaving connections are not really able to influence the throughput of well-behaving connections. Only if the connection duration of misbehaving connections increases to 1 second and higher can actual gains be made by misbehaving. When comparing the types of misbehaving TCP stacks with each other, the same observations as in Figure 3 can be made. Misbehaving TCP senders are able to achieve a higher relative gain than misbehaving TCP receivers, and the AdaptiveIgnore type of misbehaving stacks achieves a slightly higher gain than the other misbehaving TCP receivers.

In summary, these results illustrate two important points. First, all misbehaving connections do achieve a significant gain at the cost of well-behaving connections. Second, the impact of misbehaving connections is most notable when the number of concurrent connections is limited and the duration of the misbehaving connections is considerable. Misbehaving stacks can easily achieve a gain of a factor of 15. However, in other cases (fewer than 1000 connections or connection durations between 1 and 10 seconds) the relative gain is also significant. If these constraints are not met, there is no real gain in misbehaving.

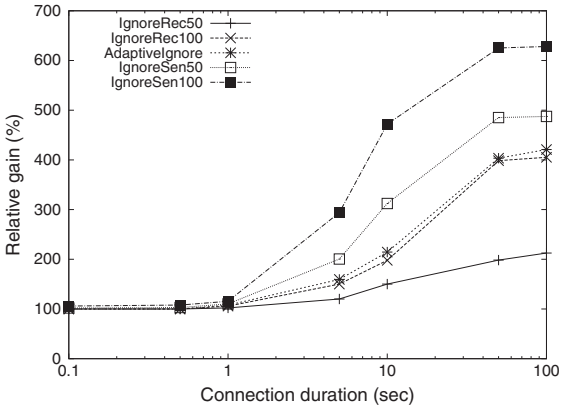

Figure 4. Relative gain of misbehaving TCP stacks given an increasing connection duration

DOI: 10.1002/nem

4. DETECTING MISBEHAVING STACKS

From the previous section it is clear that it is important to adequately respond to misbehaving TCP stacks as they can significantly decrease the fairness between connections. To do this, it is first required to detect the connections that are misbehaving. Once detected, specific countermeasures can be taken to decrease the throughput of these connections and improve the fairness. In this section, we propose a cognitive algorithm to perform such a detection. The cognitive algorithm labels incoming connections as being misbehaving, well-behaving or unknown based on flow-based statistics, collected from the traffic that passes through.

An overview of the detection algorithm and penalization component that uses the detection algorithm is illustrated in Figure 5. Based on a combination of clustering and outlier detection, the connections are divided into several clusters, called profiles. Subsequently, these profiles are labeled as being well-behaving, misbehaving or unknown. Note that, although in reality the TCP stacks that belong to this connection will misbehave or not, we focus on the labeling of misbehaving connections instead of nodes. The reason for this is, because of techniques such as NAT, we have no ability to know whether multiple TCP stacks are behind the same IP address or not. The algorithm itself is an iterative process of four steps: connection monitoring, outlier detection, clustering and labeling of misbehaving connections.

4.1. Step 1: connection monitoring

To distinguish between well-behaving and misbehaving TCP stacks, the detection algorithm extracts six statistics from each flow during a given time window W and uses them to perform the clustering and outlier detection in the subsequent steps. Through these six statistics we are able to distinguish between well-behaving and misbehaving nodes. Note that the detection algorithm monitors both upstream and downstream of connections. As such, it assumes that both directions pass through the same cognitive accountability mechanism. As our accountability mechanism is specifically targeted for a deployment in an access network this assumption holds.

- *CWR ratio*. The connection's CWR count is measured downstream (i.e. from TCP sender to TCP receiver) and denotes the number of packets containing the CWR flag in the last time window W. As such, it captures the signaling of congestion window reduced (CWR) events. The CWR count is normalized by calculating the ratio between packets containing the CWR flag and the total amount of packets received during the time window W for that connection. The CWR ratiowill mainly be influenced by misbehaving senders and to a lesser extent by misbehaving receivers. Misbehaving senders that ignore ECE flags will not lower their rate, which will result in a lower number of CWR events, captured by the CWR flags. Analogously, misbehaving receivers that ignore CE flags will trick a well-behaving or misbehaving sender to not lower its rate, which again results in a lower CWR count.
- *ECE ratio*. Similar to the CWR count, the ECE count denotes the number of packets containing the ECE flag during W. It is measured upstream (from receiver to sender) and is normalized by dividing it by the total number of upstream packets received for that connection. The main driver for a change in the ECE ratio is a misbehaving receiver. If a misbehaving receiver ignores CE flags, the CE flag will not be echoed to the sender, resulting in a drop in the ECE ratio.
- *Round trip time*. The above features allow distinguishing between well-behaving and misbehaving nodes if all flows are homogeneous. However, the heterogeneity of flows, caused by different RTTs and/or bottlenecks on locations other than the one which the algorithm is deployed on, can also lead

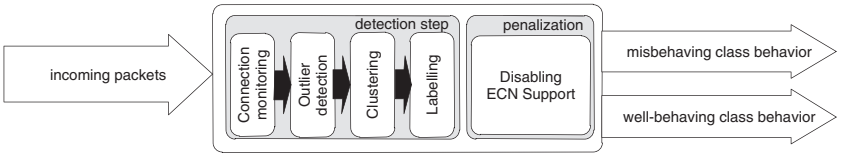

Figure 5. Overview of the detection algorithm and penalization component

DOI: 10.1002/nem

to differences in CWR and ECE ratio. The RTT is known to be inversely proportional to the obtained throughput. A higher RTT will thus lead to a lower connection throughput. To distinguish betweenthose cases, we use the RTT of a flow as a third attribute. Unlike the CWR and ECE ratio, the RTT is more difficult to characterize. To characterize this value, we use a previously proposed algorithm, called ANTMA [42], which allows estimation of the RTT of TCP connections from an intermediate node on the path between sender and receiver. We previously showed that ANTMA has an accuracy of 99.5% in calculating the RTT of a flow in random loss scenarios.

- *Connection duration*. During the start-up phase of a TCP connection, other flow statistics such as CWR and ECE ratio can only marginally characterize the connection's behavior. The connection duration is therefore also taken into account to be able to filter out these early measurements.
- *Connection rate*. As the connection rate is one of the most impacted parameters by misbehaving nodes, we also extract the rate of each connection. This value is not used for clustering but aids in labeling the clusters, as discussed in Section 4.4.
- *Average number of acknowledged packets per ACK*. This value is a configuration parameter on the TCP receiver and denotes how many consecutive packets are acknowledged with one cumulative ACK. In many cases this value will be 2. Monitoring this is easy by making a small modification to the ANTMA algorithm. Similar to the connection rate, this value is not used as a feature for clustering but is needed by the labeling algorithm.

As we will show in Section 6, this attribute set allows distinguishing between well-behaving and misbehaving nodes. Note that, although the heterogeneity of connections due to differences in RTTs is addressed by monitoring the connection's RTT using ANTMA, we are not able to distinguish between connections that suffer from a bottleneck further up or down the connection's path. This is, however, addressed in Section 4.4, where the labeling of clusters to misbehaving, well-behaving or unknown connections is discussed. Note that we monitor the connections and extract the statistics at a low frequency, i.e. once very second. This means that we do not characterize an immediate response of a TCP stack after a congestion signal, but rather investigate a more aggregate behavior that distinguishes potential misbehaving TCP stacks from other well-behaving TCP stacks. While this has the downside that it is not possible to immediately respond to misbehaving connections, this approach is considerably more scalable as a per connection management has an overhead. Moreover, the results in Section 3.4 show that short-lived connections (i.e. less than 1 second) are not able to obtain a significant gain from misbehaving.

4.2. Step 2: outlier detection

The goal of this and the following step is to divide the different flows passing through the router into different groups, called profiles. These profiles represent different behaviors that can be observed amongst the connections and are identified through an outlier detection mechanism and clustering mechanism. Both mechanisms perform their calculations based on four previously described features: RTT, connection duration, CWR ratio and ECE ratio. Thus the connection rate and number of acknowledged packets per ACK are not used as clustering features.

To ensure that a single misbehaving connection is also detected, we perform an outlier detection beforehand. This outlier detection is performed through the local outlier factor (LOF) algorithm [43], which is a density-based algorithm that is also used in data mining to detect outliers. As such, it is ideal for performing anomaly detection: the detection of a single out-of-profile behavior in a large set of connections that behave similarly.

The LOF algorithm was chosen because it has an interesting property with respect to its parameter k. Because the algorithm calculates the density with its k nearest neighbors, a cluster with $k+1$ instances will not be identified as $k+1$ outliers but as a cluster. This is illustrated in Figure 6, which shows the LOF calculation for an instance, marked X, that is part as a group of six instances (Figure 6a) and a group of five instances (Figure 6b), respectively. Note that we focus on a two-dimensional outlier detection here, while in reality our problem is four-dimensional. In both calculations, k was set to 5. As shown, the group with six instances is not considered as having outliers because its five nearest neighbors are close to each other. If one instance is removed from that group, the LOF calculation also takes into account the next

DOI: 10.1002/nem

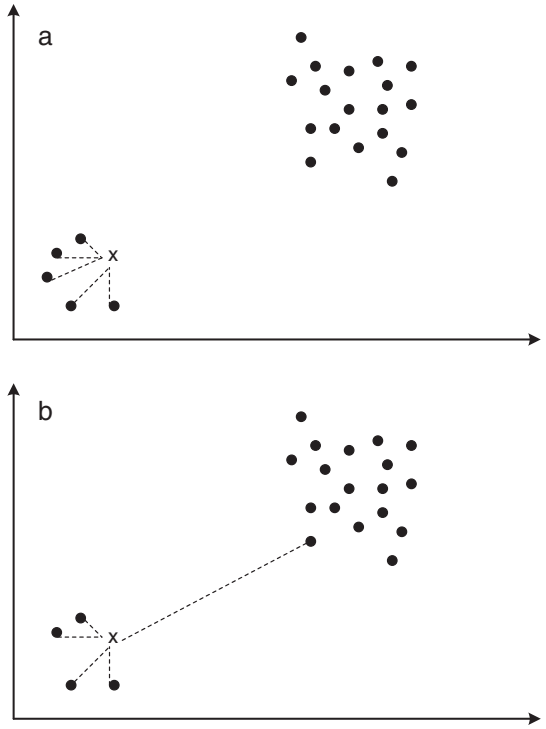

Figure 6. Graphical overview of an illustrative two-dimensional outlier calculation for a group of (a) six instances and (b) five instances. As k is set to 5, only the group of five instances are identified as outliers

nearest neighbor, which is part of a complete different group. Therefore, the LOF value is much higher and these instances are identified as outliers. In our algorithm, this characteristic is used to configure the k value in the LOF algorithm. Since an outlier can represent a misbehaving connection, it is important not to miss any outliers. Therefore, k is set to *C-1*, where C is the minimum number of instances needed in the clustering algorithm to form a cluster.

In terms of runtime complexity, the LOF algorithm consists of two sequential steps: a materialization step and an LOF value calculation step. For datasets with a low dimensionality, which is the case in our approach with only four dimensions, the runtime complexity of both steps is $O(n)$. Since the two steps are sequential, the runtime complexity of the outlier detection step is $O(n)$. We refer to Breunig *et al*. [43] for more information about the runtime complexity for higher-dimensional datasets.

4.3. Step 3: clustering

Once the outliers are identified and removed from the dataset, the remaining instances are clustered using a regular clustering mechanism, according to the four features used also in the outlier detection step. In the algorithm, each cluster is considered as a separate profile. Since it is not possible to know beforehand how many clusters remain in the dataset after the outliers are removed, we use the DBScan clustering algorithm [44] as this is one of the few algorithms that do not require the number of clusters to be defined beforehand. In terms of runtime complexity, the DBScan clustering algorithm is able to find clusters in $O(nlogn)$ time, with the aid of an indexing structure that requires $O(n^2)$ memory. Note that, since we cluster on four distinct features, it is likely that we will have a high number of clusters including the outliers that were identified from the previous step. Not all of these clusters will correspond to misbehaving profiles. It is the responsibility of the next step to accurately label these clusters as well-behaving, misbehaving or unknown.

DOI: 10.1002/nem

4.4. Step 4: labeling of misbehaving connections

The output of the previous step is a set of profiles, where it is known that each profile represents a different behavior with respect to the four clustered features. The goal of this last step is to label each connection as being either misbehaving, well-behaving or unknown, based on its membership to the current and previous clusters. The labeling algorithm that is responsible for this is illustrated in Algorithm 1. The labeling algorithm starts out with an initial labeling, performed at the previous time step *n-1* (or a labeling in which all connections are labeled as unknown during the first run). It will construct a new set of labels, denoted L_n, by calculating a value in the range between -1 and 1, where the range $\left[-1, \frac{-1}{3}\right]$ denotes a well-behaving connection, the range $\left]\frac{-1}{3}, \frac{1}{3}\right]$ denotes an unknown connection and the connections with range $\left]\frac{1}{3}, 1\right]$ are defined to be misbehaving.

To do this, the labeling algorithm first constructs an initial mapping of all profiles (not connections) to a value -1, 0 or 1 based only on the current state of the monitored flow statistics. This initial mapping is defined by the variable $\mathcal{M}$. First, profiles that have insufficient information or that is too noisy to decide on a mapping are labeled as unknown. As shown in Algorithm 1, two events can trigger this: (i) in comparing their connection's duration with the parameter ε_1, the duration is detected to be too short (lines 6–10); (ii) the standard deviation of rates (as a percentage) of the connections belonging to the profile is too high, compared to the parameter ε_2, to yield any useful information about the average rate of that cluster (lines 11–16). This can occur since we do not use the connection rate as a feature, to avoid bottlenecks on other nodes or at the sender or receiver (i.e. application-limited connections) playing a role. The impact of both ε_1 and ε_2 are evaluated in Section 2.

Algorithm 1: Algorithmic description of the labeling algorithm that labels the clustered profiles as misbehaving, well-behaving or unknown

1: Given L_n, labeling at time n
2: Set P to clustered profiles
3: $D \leftarrow \phi$
4: $M \leftarrow \phi$
5: **for all** $p \in P$ **do**
6: $d \leftarrow \sum_{i=1}^{p.size} duration(p(i))$
7: **if** $d < \varepsilon 1$ **then**
8: $P \leftarrow P \backslash p$
9: $M \leftarrow \cup \{p \rightarrow 0\}$
10: **else**
11: $std \leftarrow STDEV(rate(p(1)), \ldots, rate(p(p.size)))$
12: $r \leftarrow \sum_{i=1}^{p.size} rate(p(i))$
13: **if** $std > \varepsilon 2$ **then**
14: $P \leftarrow P \backslash \{p\}$
15: $M \leftarrow M \cup \{p \rightarrow 0\}$
16: **else**
17: $RTT \leftarrow p.centroid.RTT$
18: $b \leftarrow p.centroid.PacketACK$
19: $expected \leftarrow \frac{1}{RTT(p)\sqrt{\frac{2b}{3}} + \mathrm{T0min}(1, 3\sqrt{\frac{3b}{8}})1(1+32l2)}$
20: $D \leftarrow D \cup \{max0r - expected \rightarrow p\}$
21: **end if**
22: **end if**
23: **end for**
24: $Outliers \leftarrow LOF(D)$
25: **for all** $o \in Outliers$ **do**
26: $M \leftarrow M \cup \{o \rightarrow 1\}$
27: **end for**
28: **for all** $p \in P \backslash Outliers$ **do**

DOI: 10.1002/nem

29: $M \leftarrow M \cup \{p \rightarrow -1\}$
30: **end for**
31: **for all** $p \in P$ **do**
32: **for all** $c \in p$ **do**
33: $L_n = wL_{n-1} + (1-w)M(p)$
34: **end for**
35: **end for**

After filtering out the unknown profiles, the labeling algorithm needs to decide which profiles are misbehaving and which profiles are labeled as well-behaving. To address this, the different profiles are compared based on the rate of the cluster centroids. First, we estimate the theoretical throughput of the connections belonging to this profile based on the equation on line 19. This equation was derived from Padhye *et al*. [45], which presents a well-known model for characterizing the throughput of TCP connections in steady state. In this equation, b denotes the number of packets that are acknowledged by an ACK, T_0 is the timeout before sending unacknowledged packets (and is estimated through Paxson *et al*. [46]), and l denotes the packet loss on a link. We set l to a value close to zero to correspond to non-lossy links: this makes the calculated value *expected* an estimation of the throughput but this is, for our purposes, sufficient. Based on the calculated value *expected*, a new feature is extracted (line 20) that denotes how much the average rate of that profile exceeds the expected throughput. Note that we only take into account the rate that exceeds the expected throughput. If the rate of a profile is lower than what is theoretically expected, the calculated value for this feature will be 0. As such, this avoids rate-limited connections (e.g. because of constrained sender or receiver window sizes or because of another bottleneck) being labeled as misbehaving.

Based on the constructed $\mathcal{D}$ that contains the newly calculated feature for every profile, the LOF algorithm is again applied to detect outliers. The LOF algorithm will detect those profiles that have an abnormally high rate compared to other profiles. Detected outliers are labeled as misbehaving (lines 25–27), while the others are labeled as well-behaving (lines 28–30). Finally, the constructed labeling of profiles to misbehaving, unknown and well-behaving is smoothed and mapped to connections. To do this, an exponentially weighted moving average is calculated that, for each connection, combines the previous labeling of that connection with the novel labeling based on membership to the profile.

To derive the runtime complexity of the labeling algorithm, the algorithm can be split into three steps: the mapping of the profiles (lines 5–23), the second pass of outlier detection (lines 24–30) and the mapping of profile labels to connection labels (lines 31–35). The runtime complexity of the first step is $O(n)$, with n the number of connections, as the algorithm will need to run over all connections only once (grouped per profile). The runtime complexity of the second step is considerably less than in Section 4.2 as the LOF calculation occurs on the profiles and not on the connections. As such, it is still linear $O(n)$, but with n the number of profiles to be decided. Finally, the runtime complexity of the last step is obviously also $O(n)$ as it just runs through every connection. As such, the total complexity of the labeling algorithm is $O(n)$.

Overall, the runtime complexity of the complete detection algorithm is at most $O(nlogn)$, with n being the number of connections in the system. As the detection algorithm is executed at a low frequency of only once every second, this runtime complexity is sufficiently low to provide a scalable solution to the detection problem.

4.5. Rationale behind a cognitive approach

In essence, the cognitive algorithm labels connections as well-behaving or misbehaving based on the value of the connection statistics. This may seem a behavior that can easily be implemented by performing a simple threshold analysis on the connection statistics, which labels a connection as misbehaving if a threshold is exceeded. However, we argue that there are a number of reasons why a cognitive approach outperforms such a simple algorithm:

- *Robustness against network changes*. First, a cognitive approach is better suited against changes in the network itself. The derived connection features can easily differ depending on the context of the network scenario. If, in this case, misbehaving connections are present the detection algorithm still needs to distinguish between the two profiles. As such, the relative position of the different profiles is of importance and not the values themselves. The clustering algorithm

DOI: 10.1002/nem

takes care of this as clustering only focuses on finding groups of data and disregards the scale of the values. In an algorithm that performs threshold comparison this issue could be solved by normalizing the data. However, in this case, the existence of outliers can result in loss of information in the normalization process.

- *Robustness against connection changes*. Changes can also occur in the connections themselves. As discussed in Section 3, we varied the percentage of ECN messages that are ignored to introduce different levels of nodes misbehaving. Ignoring only a fraction of the ECN messages can be used by misbehaving connections to mask their defectiveness. In a threshold comparison algorithm, such connections could be labeled as well-behaving as they do not exceed the crisp border, defined by the threshold. The clustering algorithm allows for a smoother transition as no crisp threshold is used for dividing the connections into clusters.
- *No training set required*. A common downside of some types of cognitive approaches is that it requires to be trained to the actual scenario before deployment. As the techniques used are anomaly detection and clustering, the algorithm uses only unsupervised machine-learning techniques. This means that it does not need a training set to learn its expected behavior before deployment. Instead, a priori knowledge about the desired relative position of each profile is used to decide on the misbehaving of a profile.

5. PENALIZATION OF MISBEHAVING CONNECTIONS

Once the connections have been identified, the well-behaving and misbehaving connections are assigned to different classes, where each class receives a separate AQM behavior. We propose a simple but effective penalization action to support such a differentiated AQM behavior. The goal is to modify the AQM policy of those connections that violate the overall network fairness. Connections belonging to the well-behaving class are therefore not modified by the AQM system, as they already feature the desired behavior.

On the other hand, connections belonging to the misbehaving class are considered to be less reliable and therefore penalized. Since we focus on misbehaving TCP stacks with respect to ECN signals, the penalization consists of modifying the ECN support ofthe misbehaving class. Since connections belonging to the misbehaving class are assumed not to respond well to ECN signals, no ECN signals are sent to these connections. Hence the misbehaving flows are treated as non-ECN flows. Instead, packets that would normally be marked with a congestion experienced flag are immediately dropped from the queue. This has the effect that the misbehaving TCP stacks no longer need to react to ECN signals, and the lowering of the sending rate is immediately triggered because of packet loss. This would eventually also happen as the misbehaving of a sender or receiver leads to a continuous increase of the queue, and, in the end, to data loss as well. However, in this case, only connections belonging to the misbehaving class are affected by an increase in packet loss, while connections belonging to the well-behaving class continue to receive ECN signals. Disabling the ECN marking, i.e. treating the misbehaving connections as non-ECN flows, also has an impact on the connection's CWR and ECE count attributes. As no ECN signals are received, these ECN statistics will also be zero. Connections that are labeled as misbehaving no longer provide any useful information for the detection algorithm, and are therefore removed from the set of connections on which the misbehaving is determined by this step. Note that this approach has similarities with the earlier discussed Comcast approach [17]. However, in our case, the priority is changed by changing the ECN support. This is also in line with the recommendations made for the design of a penalty box, as originally described in the ECN specification RFC [4].

To avoid the penalization of misbehaving connections, the decision on which connections are mapped to the different classes is not solely based on the detection algorithm, detailed in the previous section. Instead, a hysteresis is introduced to avoid the oscillation of the output of the detection algorithm. The introduced hysteresis consists of an exponential back-off algorithm: if a connection is mapped as misbehaving a timer starts that lasts for T seconds. After that, the connection is labeled as unknown again. Once it is labeled as unknown, the value for the timer T is updated. Each time the detection algorithm marks the connection as misbehaving, the T value is decreased by 1 second; if the

DOI: 10.1002/nem

detection algorithm marks the connection as misbehaving again, the T value is doubled, and the connection is penalized, but this time for a longer period. New connections are thus given the benefit of the doubt, while connections that continue to give misbehaving detection results are penalized for increasingly longer periods. A good guideline for configuring the initial value of T is twice the granularity of the detection algorithm execution. Hence, as our detection algorithm runs once every second, T is initially configured at 2 seconds. This means that the labeling of a connection as misbehaving will result in the disabling of ECN marking for 2 seconds or two runs of the detection algorithm. If, after that, the labeling decision is consistent and again labeled as misbehaving, the ECN marking of the misbehaving connection will be disabled for a longer time.

Without this exponential back-off algorithm, connections that are mistakenly labeled as misbehaving would continue to be penalized, since connections with a misbehaving label are no longer considered in the detection algorithm. The exponential back-off algorithm allows one to periodically probe whether connections were not mistakenly labeled. The frequency at which this probing occurs decreases as the number of times the connection is labeled as misbehaving from the detection algorithm increases.

6. PERFORMANCE EVALUATION RESULTS

We evaluated the performance of the cognitive accountability mechanism by characterizing the accuracy of the detection algorithm on one hand and the overall gain of the penalization action on the other. As a network scenario, the network model as presented in Section 3.3 was used, consisting of two bottlenecks of which only 25% of the client terminals are affected by the 20 Mbps bottleneck. We investigated the effect of the presence of the AdaptiveIgnore, IgnoreSenX and IgnoreRecx types of misbehaving connections and varied both the share and the level of misbehaving. In all experiments, the parameter k of the outlier detection algorithm was set to 5, as the DBScan clustering algorithm was configured to interpret a group of six instances as a cluster. Each experiment was repeated 100 times: the variations between iterations were due to variations in the probabilistic models used to describe the inter-arrival time of requests and connection volume.

6.1. Detection evaluation results

As the use of penalization has an impact on the dataset used for detecting misbehaving senders and/or receivers, we disabled the penalization algorithm in this set of experiments. This allows focusing on the accuracy of the detection algorithm, avoiding any noise from errors in the detection process. As performance metrics, we focus on the precision and recall of the detection, where the labeling of misbehaving connections is seen as a positive labeling and the labeling of well-behaving connections is seen as a negative labeling. We first derive suitable values for the ε_1 and ε_2 parameters and then characterize the impact of an increasing share of misbehaving TCP stacks and level of misbehaving, as well as different RTT configurations. The RTT values for these experiments were set to $RTT_1 = 10$ ms, $RTT_2 = 30$ ms and $RTT_3 = 50$ ms, unless otherwise stated.

6.1.1. Impact of parameters

ε_1 parameter. In this experiment, we evaluate the relationship between the position of the detection algorithm in the connection and the detection accuracy. We characterize the precision and recall of labeling a connection after having monitored x seconds of the connection, in which x is varied. As such, this experiment allows one to derive a suitable value for the ε_1 parameter. To be able to investigate all values of x, we set ε_1 to zero. Furthermore, the average share of misbehaving connections was set to 30%. Figure 7 illustrates the precision and recall for an increasing x value for different types of misbehaving TCP stacks. The ε_2 parameter was set to 0.15.

Overall, we can observe an increasing precision and recall given an increasing x value. This means that increasing the ε_1 parameter will lead to an increased detection accuracy as connections which are too short or only just being monitored are labeled as unknown instead of, often incorrectly, labeled as well-behaving or misbehaving. Of course, there is a trade-off in finding this value: if we increase the ε_1 parameter, the number of connections being labeled as unknown will increase. As such, the best-suited

DOI: 10.1002/nem

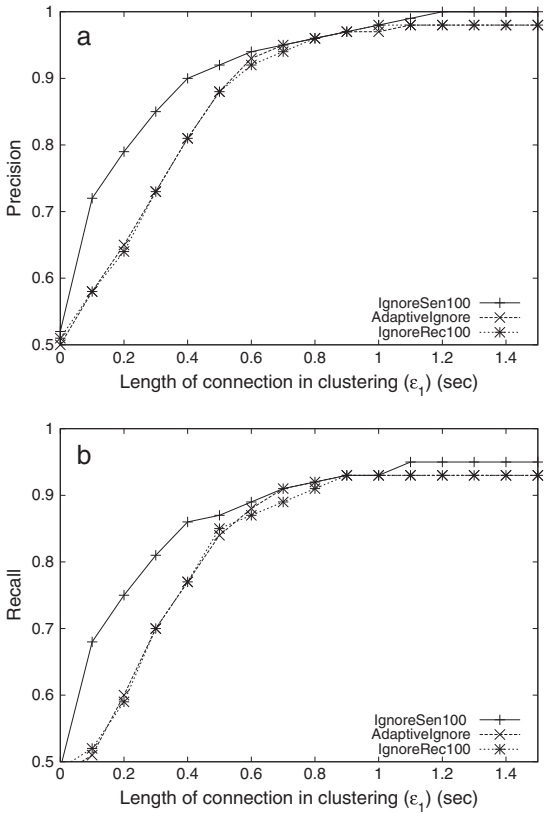

Figure 7. Precision and recall for the detection algorithm as a function of the position of the detection algorithm in the connection. This position value allows finding a well-performing ε_1 value

value for ε_1 is the smallest value that maximizes the precision and recall. As shown in Figure 7, for our experiments this is approximately 1.2 seconds. This means that connections that last longer than 1.2 seconds or are only started less than a second ago should be labeled as unknown. This is acceptable as the experiments discussed in Section 3.4 showed that no real gain can be made from misbehaving if the connections last less than a second.

When comparing the obtained precision and recall values we see that a slightly higher precision can be obtained. Hence the maximum precision value, using a $\varepsilon_1 > 1.2$ seconds, is between 98.04% and 99.87%, while the maximum recall values are between 93.12% and 95.17%. For detection algorithm accuracy this means that the algorithm almost never labels a connection as misbehaving if it is in fact well-behaving (i.e. false positives). On the other hand, the detection algorithm needs to tolerate some more false negatives, i.e. misbehaving connections that are not detected. This is an acceptable behavior as it is better to not detect misbehaving stacks than to falsely label well-behaving stacks as misbehaving.

Figure 7 also indicates that there is a difference in the detection accuracy between various types of misbehaving TCP stacks. Misbehaving senders are clearly easier to detect than misbehaving receivers. This can be seen by the higher precision and recall values for misbehaving senders earlier on in the connection. However, if the connection is monitored for a sufficiently long time, i.e. more than $\varepsilon_1 = 1.2$ seconds, both the precision and recall values for various types of misbehaving stacks converge more or less to the same value. Overall, the detection algorithm achieves a very good accuracy. For this experiment and an ε_1 parameter set to 1.2, the average precision and recall values (averaged across all types of misbehaving stacks) is 98.63% and 94.15%, respectively. In the remainder of this section, the ε_1 value was set to 1.2 seconds for all experiments as this was shown to provide good performance.

ε_2 parameter. The impact of the ε_2 parameter on the detection algorithm's performance is illustrated in Figure 8. In this figure, the precision and recall values are shown to characterize the accuracy. Additionally, we also characterize the relative share of connections with a misbehaving or well-behaving

DOI: 10.1002/nem

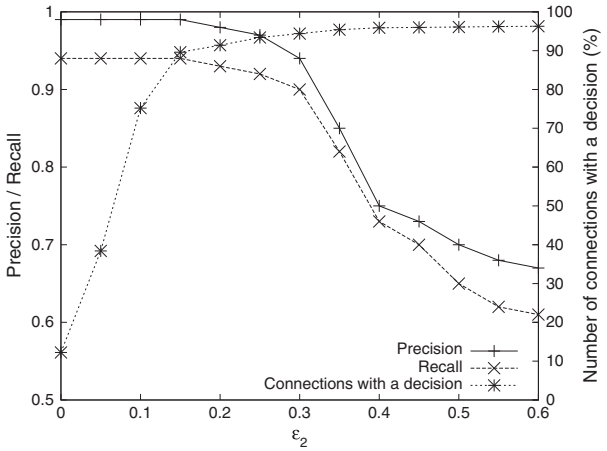

Figure 8. Impact of the ε_2 parameter in terms of precision and recall, characterizing the accuracy of the detection algorithm, and the number of connections with a decision, characterizing the connections that are not labeled as 'unknown'

label (i.e. the connections with a decision). The values are averaged across all types of misbehaving senders and receivers. We focused on senders and receivers that ignore all messages (i.e. IgnoreSen100 and IgnoreRec100) or adaptively ignore congestion warnings (i.e. AdaptiveIgnore). Each type of misbehaving sender or receiver received an equal share of the total number of misbehaving connections. Similarly to the previous experiment, the share of misbehaving connections was set to 30%.

As discussed in Section 4.4, the ε_2 parameter will tune the amount of connections that are labeled unknown because the standard deviation of the cluster's connection rate is higher than ε_2. As shown in Figure 8, the lower the value of ε_2, the better the precision and recall values seem to be. There is, however, a significant downside to a very low ε_2 value. If ε_2 is low, the number of connections that will be rejected and labeled as unknown will be very high: as such, there will be only a few connections with an actual decision. For example, with an ε_2 value set to zero, only 9.8% of the connections are labeled with a decision. This 9.8% of the connections mainly correspond to the outliers as detected by the LOF algorithm (which have a standard deviation of zero). Increasing ε_2, obviously increases the number of connections with a decision. The optimal ε_2 value must therefore satisfy the trade-off that (i) the precision and recall values are still considerably high and (ii) a sufficient number of connections are labelled with a decision. In our experiment, this value is in the range 0.15–0.20 as this is the value that corresponds to a precision and recall of approximately 98% and 94% and at the same time approximately 90% of all connections are labelled with a decision. Note that this justifies why $\varepsilon_2 = 0.15$ was chosen in the previous experiment. As shown, the number of connections with a decision slowly increases further, given an increasing ε_2, but not up to 100%. This is because connections are also labeled as unknown because of the ε_1 value, which identifies short-duration connections.

6.1.2. Impact of the share of misbehaving TCP stacks

The precision and recall of the detection algorithm as a function of the share of misbehaving TCP stacks is illustrated in Figure 9. For this experiment, we focused on the AdaptiveIgnore, IgnoreSen100 and IgnoreRec100 type of misbehaving TCP stacks (similar to the previous experiment). The labeling algorithm was configured with $\varepsilon_1 = 1.2$ and $\varepsilon_2 = 0.15$. As shown, the recall is only marginally influenced by an increasing share of misbehaving TCP stacks. With the exception of a small drop in recall when only 10% of connections are misbehaving, the recall remains stable at 94.15%. The precision is also fairly stable, although somewhat more impacted by the lower share of misbehaving connections. As shown, the precision suffers from a drop in the region of 10–30%. However, overall, the detection algorithm remains very accurate, with precision values in the range between 82.15% (10% of misbehaving connections) and 98.63% (40% of misbehaving connections and more).

DOI: 10.1002/nem

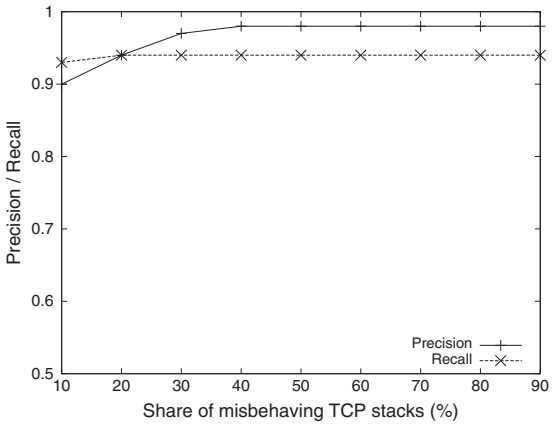

Figure 9. Precision and recall for an increasing share of misbehaving TCP stacks

6.1.3. Impact of the level of misbehaving TCP stacks

In this experiment, we investigate the impact of an increasing level of misbehaving TCP stacks. We focus only on the precision of the algorithm as the recall has been shown to be very stable (similar to the previous results). For this experiment, 30% of all connections were misbehaving, of which 50% were misbehaving senders and the remainder were misbehaving receivers. Figure 10 investigates the precision values for both types of misbehaving stacks as a function of the percentage of ignored congestion warnings. As shown, the best precision can be obtained when misbehaving TCP stacks ignore more than 50% of all congestion warnings. In this case, a precision of 99.87% and 98.04% can be achieved in detecting misbehaving senders and receivers, respectively. If only a small fraction of the congestion warnings are ignored, a considerable drop in precision is experienced. For example, when only 10% of the warnings are ignored, 87.12% of all misbehaving receivers can be detected, while 92.54% of all misbehaving senders can be detected.

In our opinion, this result is still acceptable for two reasons. First, although we perceive a drop in precision, the drop is notable but still limited. On average, approximately nine out of ten misbehaving connections can be detected. Additionally, the precision increases very quickly if more congestion warning messages are ignored. Second, the gain that can be achieved by ignoring only a small fraction of the congestion warnings is very limited. The results in Section 3.4 have already shown that ignoring fewer congestion warnings results in a reduction in the gain that can be achieved by misbehaving. For example, for connections with an average duration of 10 seconds, ignoring half of the congestion warnings, results in a gain which is only half of when all congestion warnings are ignored. For lower percentages of ignored congestion warnings, this gain is even considerably lower. For example, if 10% of all congestion warnings are ignored, the achieved gain is less than 10% compared to well-behaving

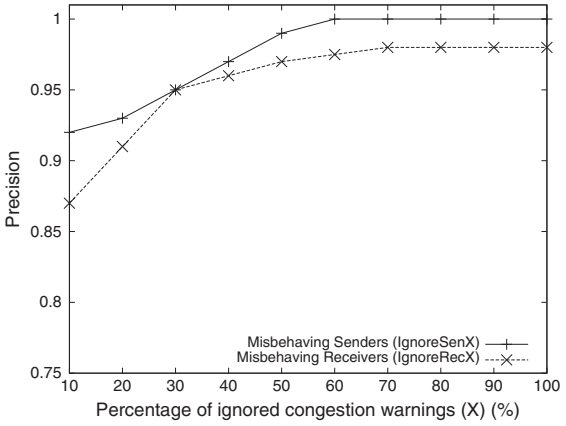

Figure 10. Precision and recall for an increasing level of misbehaving TCP stacks

DOI: 10.1002/nem

stacks. As such, the impact of these misbehaving TCP stacks on the overall fairness between connections is limited. Hence the issue of not detecting one out of ten misbehaving connections is less severe.

6.1.4. Impact of differences in RTT

As the obtained throughput of a connection is inversely proportional to the RTT of that connection, there is an important link between the fairness of connections and their RTT. In the detection algorithm, we use the estimated RTT value of the ANTMA algorithm as an attribute in the clustering and outlier detection process. As such, the detection algorithm is able to cope with differences in RTT amongst connections. Since already different RTT values were configured for all three sites in the experimental setup, the previous results have already confirmed that a high precision and accuracy can be obtained, even if differences exist in the RTT between connections. In this section, we evaluate the stability of these precision and recall values for different RTT configurations.

Table 1 characterizes both the precision (P) and recall (R) for different types of misbehaving TCP stacks and under three different configurations of RTT values for the three sites in the experimental setup. As shown, the results do not diverge much across different RTT configurations and confirm the earlier observations in the previous sections. For all RTT values, we observe a higher precision value than recall value, resulting in slightly more false negatives than false positives. Overall, the precision and recall values are high and do not change considerably under varying RTT configurations. This shows that the accuracy of the detection algorithm is stable under varying RTT configurations. Although the difference is limited, there is a small increase in accuracy when the RTT values do not diverge much. Indeed, the results in the columns 3 and 4 show a slightly higher precision and recall value compared to the other configuration. This can be expected as the problem of accurately detecting misbehaving connections is reduced if there is no real difference between the obtained RTT values.

6.2. Penalization evaluation results

While the previous section focused on an offline detection accuracy evaluation, in this section we characterize the gain of the complete system, including the penalization action proposed in Section 5. Therefore, the detection algorithm was deployed online and its output was provided to the penalization algorithm, which includes the exponential back-off algorithm with T initially set to 5 seconds. The same experimental setup configuration was used as investigated in Section 3.4. The connection-monitoring step generated an updated statistics value every second and also triggered the detection algorithm and, ultimately, the penalization action. We investigate the impact of applying penalization to the detected connections by characterizing the gain that can be achieved under the same conditions as discussed in Section 3.4.

6.2.1. Number of concurrent connections

Figure 11 compares the gain achieved by misbehaving TCP stacks as a function of the number of concurrent connections. Without penalization, we observe the same results as discussed in Section 3.4: a relative gain of up to 1541.19% can be achieve by misbehaving, which diminishes as the number of concurrent connections increases. As shown in Figure 11, applying penalization to the detected misbehaving TCP stacks successfully reduces this relative gain. In all cases, the relative gain achieved by penalized connections is close to 100%. Note that 100% means that there is no difference between

Table 1. Precision (P) and recall (R) values for different types of misbehaving TCP stacks under various RTT configurations (ms) for the three investigated sites

	$RTT_1 = 30$, $RTT_2 = 30$ $RTT_3 = 30$		$RTT_1 = 10$, $RTT_2 = 30$ $RTT_3 = 50$		$RTT_1 = 10$, $RTT_2 = 50$ $RTT_3 = 100$	
	P	R	P	R	P	R
IgnoreSen100	99.98%	98.42%	99.87%	95.17%	98.11%	94.18%
AdaptiveIgnore	99.01%	97.13%	98.14%	93.46%	97.82%	92.87%
IgnoreRec100	98.78%	97.13%	98.04%	93.12%	97.78%	92.59%

DOI: 10.1002/nem

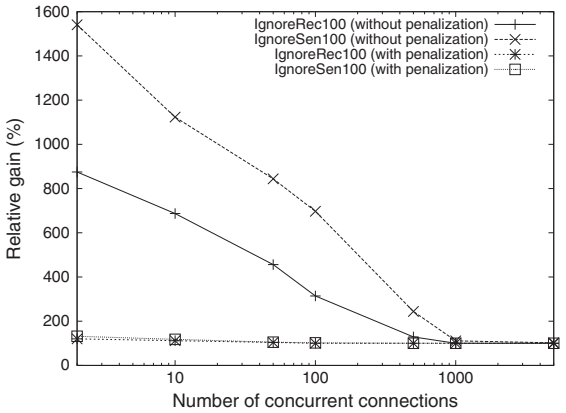

Figure 11. Gain achieved by misbehaving TCP senders and receivers with and without penalization as a function of the number of concurrent connections

misbehaving and well-behaving connections: a relative gain below 100% means that misbehaving connections experience a lower throughput than the well-behaving connections. Although the relative gain is close to 100% for a small number of concurrent connections, applying penalization cannot completely avoid misbehaving connections achieving some gain in misbehaving. For example, penalized misbehaving senders and receivers achieve an average gain of 132.52% and 120.78%, respectively. This is, however, a considerable reduction of the achieved relative gain for misbehaving senders and receivers without penalization (1014.57% and 1541.19%, respectively). As the number of concurrent connections increases, the achieved relative gain of penalized misbehaving connections is diminished. For 50 concurrent connections, the average relative gain is 105.25% and 104.17% for misbehaving senders and receivers, respectively. For 100 concurrent connections and more, no additional gain can be achieved by misbehaving. When comparing the gain between penalized misbehaving senders and receivers, we see that they have a similar but not equal relative gain. Similar to the unpenalized case, misbehaving senders achieve a consistently higher gain than misbehaving receivers. However, because of penalization, the difference in gain can be reduced to a few percent.

6.2.2. Connection duration

The comparison between unpenalized and penalized misbehaving connections as a function of the connection duration is shown in Figure 12. A similar behavior can be observed to that discussed in the previous section. Applying penalization allows a considerable but not complete reduction of the relative gain. Especially if unpenalized connections achieve a very high gain, i.e. more than 500%,

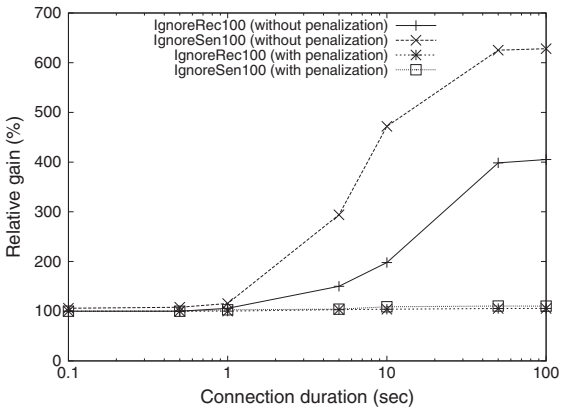

Figure 12. Gain achieved by misbehaving TCP senders and receivers with and without penalization as a function of the connection duration

DOI: 10.1002/nem

the penalizing system needs to tolerate a limited gain from misbehaving connections. For example, for connections with an average duration of 100 seconds, the relative gain of misbehaving senders and receivers is 110.25% and 105.55%, respectively. These are the highest gains of penalized misbehaving connections that could be observed and, for shorter-lived connections, the relative gain is considerably lower. For example, for connections with an average duration of 10 seconds, the relative gain values that were obtained were 104.84% and 104.05% for misbehaving senders and receivers, respectively. Overall, the system is able to achieve a very good performance by almost completely reducing the difference between misbehaving and well-behaving connections.

7. CONCLUSIONS

In this article we presented a cognitive accountability mechanism that is able to restore the fairness between TCP connections, originally affected by the presence of misbehaving TCP senders or receivers that (partially) ignore ECN congestion signals. The cognitive accountability mechanism is deployed on top of a normal AQM-enabled router and consists of a detection algorithm and differentiated AQM mechanism. The detection algorithm monitors each connection passing through the router and performs a combination of clustering and outlier detection based on the monitored data to detect different behavioral groups, called profiles. The detection mechanism comprises a labeling algorithm that exploits the statistics that were extracted from each profile to decide whether or not the connections belonging to that profile are misbehaving, well-behaving or (temporarily) unknown. The output of the detection algorithm is used to perform a differentiated AQM behavior that divides connections into a well-behaving and misbehaving class and penalizes the misbehaving class by disabling the ECN marking. We showed that misbehaving connections can mainly achieve a considerable gain in network environments when (i) they have a considerable connection duration (i.e. more than 1 second) and (ii) a moderate number of concurrent connections or fewer exist (i.e. a few thousand concurrent connections or fewer). Therefore, the proposed cognitive accountability is mainly targeted for deployment in a multimedia access network, preferably positioned close to the receiver. Note that the approach does not immediately suffer from scalability issues if these constraints are not met, but that there is less gain to be made in misbehaving.

The cognitive accountability mechanism has been evaluated on a simulation environment emulating a multimedia access network with multiple bottlenecks and three different remote sites, having different RTT configurations. Different evaluation scenarios were introduced by varying the type of misbehaving connections, the share of misbehaving connections, the level of misbehaving and the network configuration itself. Both the accuracy of the detection algorithm and the achieved fairness gain of the overall accountability mechanism have been characterized. The key findings of the performance evaluation are the following. First, we showed that misbehaving senders and receivers can be detected very accurately in the network environments where they achieve an actual gain of misbehaving. In these cases, the detection algorithm has a precision of 97% or more, and a recall of 92% or more. Second, because of a careful selection of clustering features, the detection algorithm is robust against changes in the network configuration such as rate limits at the receiver or sender side or differences in the RTT. We showed that differences in RTT between sites has an impact on the accuracy but that this impact is limited (i.e. of the order of 2%). Third, the results show that misbehaving senders are able to achieve a higher gain than misbehaving receivers but that they are also easier to detect. Fourth, applying penalization to the misbehaving connections allows diminishing the achieved gain by misbehaving almost completely but not fully. The results show that a gain of up to 844% can be decreased to only 105%, which means that misbehaving connections have an average throughput which is only 5% higher than well-behaving connections. In some cases, we are able to completely equalize the throughput of both classes of connections. Overall, we have shown that the proposed cognitive accountability mechanism is able, accurately and in a timely manner, to detect the occurrence of misbehaving senders and receivers. Moreover, the penalization of the misbehaving connections does indeed lead to a better overall network fairness. In future work, we plan to investigate other types of misbehaving TCP stacks, not necessarily focusing on misbehaving ECN behavior.

DOI: 10.1002/nem

ACKNOWLEDGEMENTS

Steven Latré is funded a by grant from the Fund for Scientific Research, Flanders (FWO-V). Dirk Deschrijver is funded by a postdoctoral research grant from FWO-V. The research was performed partly within the framework of the FP7 STREP ECODE project.

REFERENCES

1. Sadre R, Haverkort B. Changes in the web from 2000 to 2007. In *Managing Large-Scale Service Deployment: Proceedings of the International Workshop on Distributed Systems: Operations and Management (DSOM)*. Lecture Notes in Computer Science, Vol. **5273**. Springer: Berlin, 2008; 136–148.
2. Papadimitriou D, Welzl M, Scharf M, Briscoe B. Open research issues in Internet congestion control. *RFC 6077*, January 2011. Available: http://www.ietf.org/rfc/rfc6106.txt [11 March 2012].
3. Sherwood R, Bhattacharjee B, Braud R. Misbehaving tcp receivers can cause Internet-wide congestion collapse. In *Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS '05*. ACM: New York, 2005; 383–392.
4. Ramakrishnan K, Floyd S, Black D. The addition of explicit congestion notification (ECN) to IP. *RFC 3168* (Proposed Standard), September 2001. Available: http://www.ietf.org/rfc/rfc3168.txt [11 March 2012], updated by *RFCs 4301*, 6040.
5. Jacobson V. Congestion avoidance and control. *SIGCOMM Computer Communication Review* 1988; **18**: 314–329.
6. Chiu DM, Jain R. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems* 1989; **17**(1): 1–14.
7. Allman M, Paxson V, Stevens W. TCP congestion control. *RFC 2581* (Proposed Standard) April 1999. Available: http://www.ietf.org/rfc/rfc2581.txt [11 March 2012],, obsoleted by *RFC 5681*, updated by *RFC 3390*.
8. Floyd S, Henderson T. The NewReno modification to TCP's fast recovery algorithm. *RFC 2582* (Experimental), April 1999. Available: http://www.ietf.org/rfc/rfc2582.txt [11 March 2012], obsoleted by *RFC 3782*.
9. Floyd S, Jacobson V. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking* 1993; **1**(4): 397–413.
10. Feng WC, Shin KG, Kandlur DD, Saha D. The blue active queue management algorithms. *IEEE/ACM Transactions on Networking* 2002; **10**: 513–528.
11. de Veciana G, Konstantopoulos T, Lee TJ. Stability and performance analysis of networks supporting elastic services. *IEEE/ACM Transactions on Networking* 2001; **9**: 2–14.
12. Zhu X, Yu J, Doyle J. Heavy tails, generalized coding, and optimal web layout. INFOCOM 2001: Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. *Proceedings of the IEEE* 2001; **3**: 1617–1626.
13. Kelly F, Maulloo A, Tan D. Rate control in communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society* 1998; **49**: 237–252.
14. Savage S, Cardwell N, Wetherall D, Anderson T. Tcp congestion control with a misbehaving receiver. *SIGCOMM Computer Communication Review* 1999; **29**: 71–78.
15. Spring N, Wetherall D, Ely D. Robust explicit congestion notification (ECN) signaling with Nonces. *RFC 3540* (Experimental), June 2003. Available: http://www.ietf.org/rfc/rfc3540.txt [11 March 2012].
16. Kulatunga C, Fairhurst G. Enforcing layered multicast congestion control using ecn-nonce. *Computer Networks* 2010; **54**(3): 489–505.
17. Bastian C, Klieber T, Livingood J, Mills J, Woundy R. Comcast's protocol-agnostic congestion management system. *RFC 6057* (Informational), December 2010. Available: http://www.ietf.org/rfc/rfc6057.txt [11 March 2012].
18. Kabbani A, Prabhakar B. *defense of TCP. The Future of TCP: Train-wreck or Evolution*. Stanford University: Stanford, CA, 2008.
19. Alizadeh M, Greenberg A, Maltz DA, Padhye J, Patel P, Prabhakar B, Sengupta S, Sridharan M. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference on SIGCOMM*. ACM: New York, 2010; 63–74.
20. Moncaster T, Leslie J, Briscoe B, Woundy R, McDysan D. Conex concepts and use cases. Internet Draft draft-ietfconex- concepts-uses-01, Internet Engineering Task Force, March 2011.
21. Mathis M, Briscoe B. Congestion exposure (conex) concepts and abstract mechanism. Internet Draft draft-ietfconex-abstract-mech-01, Internet Engineering Task Force March 2011.
22. Briscoe RB. Re-feedback: freedom with accountability for causing congestion in a connectionless Internetwork. PhD thesis, UC London, 2009. Available: http://www.bobbriscoe.net/projects/refb/refb_dis.pdf [12 March 2012].
23. Briscoe B, Jacquet A, Moncaster T, Smith A. Re-ECN: adding accountability for causing congestion to TCP/IP. Internet Draft draft-briscoe-tsvwg-re-ecn-tcp-09.txt, Internet Engineering Task Force, October 2010.
24. Briscoe B, Jacquet A, Di Cairano-Gilfedder C, Salvatori A, Soppera A, Koyabe M. Policing congestion response in an internetwork using re-feedback. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for computer Communications, SIGCOMM '05*. ACM: New York.
25. Briscoe B. A fairer, faster internet protocol. *IEEE Spectrum* 2008; **December**: 38–43.
26. Huang X, Wu J, Sun G, Jing J. A new fair active queue management algorithm. In *2009 International Conference on Future Networks*, 2009; 191–195.
27. Hanlin S, Yuehui J, Yidong C, Hongbo W, Shiduan C. Improving fairness of red aided by lightweight flow information. In *2nd IEEE International Conference on Broadband Network Multimedia Technology*; 335–339.

DOI: 10.1002/nem

28. Eardley P. Pre-Congestion notification (PCN) architecture. *RFC 5559* (Informational), June 2009. http://www.ietf.org/rfc/rfc5559.txt [11 March 2012].
29. Kuzmanovic A, Knightly EW. Receiver-centric congestion control with a misbehaving receiver: vulnerabilities and end-point solutions. *Computer Networks* 2007; **51**: 2717–2737.
30. Ekiz N, Rahman AH, Amer PD. Misbehaviors in tcp sack generation. *SIGCOMM Computer Communication Review* 2011; **41**: 16–23.
31. Cheng RS, Lin HT. Protecting tcp from a misbehaving receiver. *International Journal of Network Management* 2007; **17**(3): 209–218.
32. Chan ACF. Efficient defence against misbehaving tcp receiver dos attacks. *Computer Networks* 2011; **55**: 3904–3914.
33. Hu T, Fei Y. Qelar: a machine-learning-based adaptive routing protocol for energy-efficient and lifetime-extended underwater sensor networks. *IEEE Transactions on Mobile Computing* 2010; **9**(6): 796–809.
34. van den Berg E, Fecko MA, Samtani S, Lacatus C, Patel M. Distributed game-theoretic topology control in cognitive networks. *Proceedings of SPIE* 2010; **77** 070E.
35. Song J, Takakura H, Okabe Y, Inoue D, Eto M, Nakao K. A comparative study of unsupervised anomaly detection techniques using honeypot data. *IEICE Transactions on Information and Systems* 2010; **E93.D**(9): 2544–2554.
36. Brauckhoff D, Salamatian K, May M. A signal processing view on packet sampling and anomaly detection. In *IEEE Infocom*, March 2010.
37. Raineri F, Verticale G. Early Internet application identification with machine learning techniques. In *First International Conference on Evolving Internet: INTERNET '09*, 2009; 60–64.
38. El Khayat I, Geurts P, Leduc G. Enhancement of tcp over wired/wireless networks with packet loss classifiers inferred by supervised learning. *Wireless Networks* 2010; **16**: 273–290.
39. Jayaraj A, Venkatesh T, Murthy C. Loss classification in optical burst switching networks using machine learning techniques: improving the performance of tcp. *IEEE Journal on Selected Areas in Communications* 2008; **26**(6): 45–54.
40. Ns-2, The Network Simulator. Available: http://www.isi.edu/nsnam/ns/ [11 March 2012.
41. Pustisek M, Humar I, Bester J. Empirical analysis and modeling of peer-to-peer traffic flows. In *14th IEEE Mediterranean Electrotechnical Conference (MELECON'08)*, 2008; 169–175.
42. De Vleeschauwer B, Van De Meerssche W, Simoens P, De Turck F, Dhoedt B, Demeester P, Van Caenegem T, Dequeker H, Struyve K, Gilon E, *et al*. Enabling autonomic access network QoE management through TCP connection monitoring. In *Proceedings of the 1st IEEE Workshop on Autonomic Communications and Network Management (ACNM'07)*, 2007.
43. Breunig MM, Kriegel HP, Ng RT, Sander J. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD '00)*. ACM: New York, 2000; 93–104.
44. Ester M, Kriegel HP, Sander J, Xu X. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, Vol. **96**. AAAI Press: Palo Alto, 1996; 226–231.
45. Padhye J, Firoiu V, Towsley D, Kurose J. Modeling tcp throughput: a simple model and its empirical validation. *SIGCOMM Computer Communication Review* 1998; **28**: 303–314.
46. Paxson V, Allman M, Chu J, Sargent M. Computing TCP's retransmission timer. *RFC 6298* (Proposed Standard), June 2011. Available: http://www.ietf.org/rfc/rfc6298.txt.

AUTHORS' BIOGRAPHIES

Steven Latré obtained a master's degree in computer science from the University of Ghent, Belgium, in June 2006. In August 2006, he joined the Department of Information Technology at the University of Ghent, where he received a PhD in engineering: computer science in June 2011. His main research interest is the use of autonomic network management approaches with a special focus on quality of experience optimization and management of federations. He is an author of more than 35 conference and/or journal publications. He was also involved in the IST FP6 project MUSE, EUREKA CELTIC project RUBENS, and FP7 STREP ECODE project and is currently participating in the FP7 STREP OCEAN project as well as several other, national, projects.

Wim Van de Meerssche received his MSc degree in software development in 2004 from the University of Ghent, Belgium. In August 2004 he started working on software technologies for access networks in the Department of Information Technology (INTEC) at the same university. His work has been published in several scientific publications in international conferences. He was involved in the IST FP6 project MUSE, EUREKA CELTIC project RUBENS and is currently involved in the FP7 STREP project ECODE.

Dirk Deschrijver received the master's degree (Licentiaat) in computer science and PhD degree from the University of Antwerp, Belgium, in 2003 and 2007, respectively. He was with the Computer Modeling and Simulation (COMS) Group, University of Antwerp, where he was supported by a research project of the Fund for Scientific Research Flanders (FWOVlaanderen). From May to October 2005 he was a Marie Curie Fellow with the Scientific Computing Group, Eindhoven University of Technology, The Netherlands. He is currently

DOI: 10.1002/nem

an FWO Post-Doctoral Research Fellow with the Department of Information Technology (INTEC), University of Ghent, Belgium. His research interests include rational least squares approximation, orthonormal rational functions, system identification and parametric macromodeling techniques.

Dimitri Papadimitriou joined the Network Architecture team of the Central Research Center of the Alcatel CTO in 2000 as Expert Research Engineer, where he was in charge of the multi-layer packet/optical network architecture. Since 2006 he has worked as Principal Research Engineer on Future Internet research thematics for Alcatel-Lucent Bell Labs. His current areas of investigation are focused on Internet architecture and formal design methodologies, topology modeling and graph analysis/mining, distributed algorithms for dynamic routing, as well as application of learning paradigms to communication networks. He is currently leading two EU Framework Program 7 (FP7) projects on Future Internet, including the ECODE project, which combines machine learning and advanced networking research, and the EIFFEL Project. He is also actively involved in the standardization activities of the Internet Research Task Force (Routing and Congestion Control Research Groups), and Internet Engineering Task Force (Routing and Internet Area).

Tom Dhaene received a PhD degree in electrotechnical engineering from the University of Ghent, Belgium, in 1993. From 1989 to 1993 he was a Research Assistant with the Department of Information Technology, University of Ghent, where his research focused on different aspects of full-wave electromagnetic (EM) circuit modeling, transient simulation, and time-domain characterization of high-frequency and high-speed interconnections. In 1993 he joined the EDA company Alphabit (now part of Agilent Technologies). He was one of the key developers of the planar EM simulator ADS Momentum, and he is theprincipal developer of the multivariate EM-based adaptive metamodeling tool ADS Model Composer. He was a Professor with the Computer Modeling and Simulation (COMS) Group, Department of Mathematics and Computer Science, University of Antwerp, Belgium. He is currently a Full Professor with the Department of Information Technology, University of Ghent.

Filip De Turck received his MSc degree in electronic engineering from the University of Ghent, Belgium, in June 1997. In May 2002 he obtained a PhD degree in electronic engineering from the same university. He is currently a full-time professor affiliated with the Department of Information Technology of the University of Ghent and the IBBT (Interdisciplinary Institute of Broadband Technology Flanders) in the area of telecommunication and software engineering. He is the author or co-author of approximately 200 papers published in international journals or in the proceedings of international conferences. His main research interests include scalable software architectures for telecommunication network and service management.

DOI: 10.1002/nem