Gaining Control of Cellular Traffic Accounting by Spurious TCP Retransmission

Younghwan Go
KAIST
yhwan@ndsl.kaist.edu

Jongil Won
KAIST
wji4@ndsl.kaist.edu

Denis Foo Kune
University of Michigan
foo@eecs.umich.edu

EunYoung Jeong
KAIST
notav@ndsl.kaist.edu

Yongdae Kim
KAIST
yongdaek@kaist.ac.kr

KyoungSoo Park
KAIST
kyoungsoo@ee.kaist.ac.kr

***Abstract*—Packet retransmission is a fundamental TCP mechanism that ensures reliable data transfer between two end nodes. Interestingly, when it comes to cellular data accounting, TCP retransmissions create an important policy issue giving rise to a tension between ISPs accounting for network resource consumption, and users only being aware of the application layer data. Regardless of the policies, we find that TCP retransmissions can be easily abused to manipulate the current practice of cellular traffic accounting.**

In this work, we investigate the TCP retransmission accounting policies of 12 cellular ISPs in 6 countries and report the accounting vulnerabilities with TCP retransmission attacks. First, we find that cellular data accounting policies vary between ISPs. While the majority of cellular ISPs blindly account for every IP packet, some ISPs intentionally remove the retransmission packets from the user bill for fairness. Second, we show that it is easy to launch the "usage-inflation" attack on the ISPs that blindly account for every IP packet. In our experiments, we could inflate the usage up to the monthly limit with an attack invisible to the subscriber and lasting only 9 minutes. For those ISPs that do not account for retransmission, we successfully launch the "free-riding" attack by tunneling the payload over fake TCP headers that look like retransmissions. To counter the attacks, we implement and evaluate Abacus, a light-weight, scalable accounting system that reliably detects "free-riding" attacks even in the 10 Gbps links.

I. Introduction

The demand for cellular Internet access is rapidly increasing, reaching over 2 billion mobile broadband subscribers worldwide in 2013 [1], making accurate accounting of cellular traffic all the more important. Most cellular Internet Service Providers (ISPs) adopt byte-level accounting of consumed IP packets that flow through their cellular networks [1]. Typical access plans limit the monthly usage to a few GBs, and the subscribers pay high overage fees when their data consumption exceeds the pre-negotiated monthly limit.

Byte-level usage accounting, however, presents an important policy issue when it comes to TCP traffic. For reliable data transfer, TCP automatically retransmits the packets when a sender receives 3 duplicate ACKs [2], [3] or Retransmission Timeout (RTO) triggers [4]. Cellular traffic accounting systems must decide whether to account for those retransmitted TCP packets. Given that over 95% of the cellular traffic is based on TCP [5], [6], the accounting policy could affect the majority of cellular traffic in practice.

One approach is to charge for every IP packet regardless of TCP packet retransmissions. At a glance, this "blind accounting" looks to be a reasonable choice to cellular ISPs since every IP packet consumes resources in their infrastructure. It is not only simple to implement but overcharging due to packet retransmission is expected to be small in normal situations since typical packet retransmission rates are low (0.4 to 1.7%) in well-provisioned cellular infrastructures. However, this policy exposes the cellular subscribers to a "usage-inflation" where an adversary arbitrarily inflates the usage by intentionally retransmitting packets from remote servers. Unfortunately, it is non-trivial to reliably distinguish between valid and fake retransmissions since our measurements indicate that retransmission rates can go up to from 40% to 85% in poorly-provisioned areas.

The alternative approach is to remove retransmitted packets from the usage amount for billing. The rationale behind this "selective accounting" is to hold the ISP responsible for any retransmissions since the wireless cellular architecture is more likely to drop packets, particularly in those poorly-provisioned areas. This model also fits well with subscribers' conceptual view that they should be charged only for application data without penalties from the underlying transport layer. While this avoids the "usage-inflation" attack, it significantly increases the implementation complexity by imposing the flow

NDSS '14, 23-26 February 2014, San Diego, CA, USA

http://dx.doi.org/10.14722/ndss.2014.23118

[1] Flat-fee plans are available for 3G (or lower-speed) access in some ISPs but unrestricted flat-fee plans are rare for 4G or LTE access.

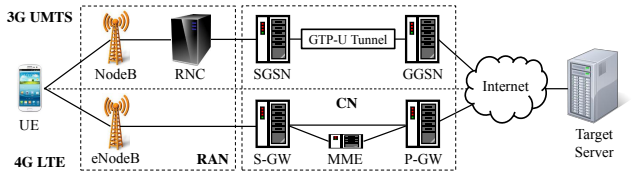


Fig. 1. Overall architecture of UMTS/LTE cellular network

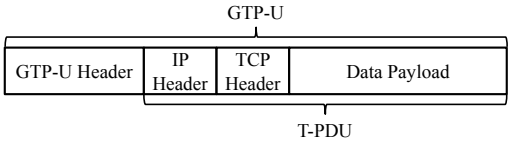


Fig. 2. GPRS packet format between the GSNs

management burden on the accounting system. In addition, improper implementation could be vulnerable to "free-riding" attacks if attackers tunnel the actual content over fake TCP headers masquerading as TCP retransmission packets.

In this paper, we explore possible attacks on cellular accounting systems with TCP retransmissions. We first investigate the accounting policies of 12 cellular ISPs around the world. We find that 9 cellular ISPs blindly account for every IP packet, but none of them defend against the "usage-inflation" attack with malicious TCP retransmission nor have informed us of the attack existence. Our experiments show that an attacker can increase the usage of an arbitrary subscriber to their monthly limit (in our case, 1GB) with an attack lasting only a few minutes. We also find that some ISPs leave connections open indefinitely making those vulnerable to usage-inflation attacks despite repeated client-side closures. The remaining 3 ISPs carefully take out the retransmission packets from the bill, but we observe that their current implementation is vulnerable to the "free-riding" attack. We have implemented tunneling proxies and protocols for the attack, and show that one can use the cellular data service for free at the speed of 15.6 to 22.1 Mbps with the ISPs [2].

There have been some reports of cellular accounting vulnerabilities due to simple policy loopholes [7], [8] or packet drops from service roaming [9]. In contrast, detecting or preventing the attacks that exploit TCP retransmission bears the *fundamental* difficulty tied to the reliability mechanism of the TCP protocol itself. The core problem lies in that TCP is operated on end nodes while the accounting middleboxes deal with the IP layer in the middle of the network. Specifically, middleboxes do not know the state of each TCP context and cannot reliably infer whether a packet is retransmitted out of necessity or out of malicious will. The problem is similar to detecting aggressive TCP senders that ignore congestion control in the routers, which is known to be difficult [10]–[12].

To work around these attacks, we propose that the cellular ISPs should remove TCP retransmission packets from the user bill, but they should develop a robust accounting system that detects the "free-riding" attacks by flow-level Deep Packet Inspection (DPI). To hint that such a system is possible, we show the design and implementation of *Abacus*, a light-weight byte-level accounting system that manages hundreds of thousands of concurrent connections and runs probabilistic DPI on the payload that reliably catches the "free-riding" attack on a commodity machine.

The rest of the paper is organized as follows. In Section II, we explain the overall architecture and the accounting process of cellular data traffic, and discuss the causes for TCP retransmission in the cellular infrastructure. We investigate the accounting policies of various ISPs, and compare the packet retransmission rates in cellular TCP traffic in Section III. In Section IV, we implement the "usage-inflation" and "free-riding" attacks and check how current ISPs behave against them. We propose an accurate accounting system that detects the "free-riding" attacks in Section V and evaluate the system in Section VI. Finally, we look at previous works in cellular traffic characteristics and accounting in Section VII, and conclude in Section VIII.

II. Background

In this section, we describe the architecture of the cellular network and the accounting process of cellular data traffic. We mainly focus on the Universal Mobile Telecommunications System (UMTS) [13] for 3G and Long Term Evolution (LTE) [14] for 4G. The architecture is based on a Packet-Switched (PS) domain, in which the data is transferred in packets [15]–[18]. We then look at the possible causes of TCP packet retransmission in cellular networks.

A. Cellular Network Architecture

The overall cellular network architecture is divided into two parts: (1) the wireless links between the client and the cellular ISP, and (2) the wired links between the cellular ISP and the target server (Figure 1). When a user initiates a connection to the cellular network (e.g., turning on the cellular interface of a device), their User Equipment (UE) first creates a channel between a base station (NodeB for 3G, eNodeB for LTE) located in the Radio Access Network (RAN). In the case of 3G networks, the NodeB is connected to a Radio Network Controller (RNC), which controls the NodeB operations such as handling the message encryption and allocating the radio resources. It is also responsible for managing the UE's mobility such as handling soft handovers between two RANs, enabling the UE to connect to another NodeB at different RAN to provide a seamless communication. For LTE, the eNodeB incorporates the RNC functions.

Once the UE's packets arrive at the RAN, they are transferred via a wired network. The cellular ISP's Core Network (CN) receives the packets and processes them via the General Packet Radio Service (GPRS). The packet processing is done as follows. (1) All packets transmitted from the RAN are received and logged for billing purposes by the Serving GPRS Support Node (SGSN) or Serving Gateway (S-GW). In the case of LTE, the Mobility Management Entity (MME) is responsible for tracking the UE and choosing the appropriate S-GW to serve the UE. It is also responsible for mobility between 3G and LTE. At this point, all packets are carried around the CN in the form of GPRS Tunneling Protocol (GTP-U [19]) packets as illustrated in Figure 2. The GTP-U contains

[2]We have reported the vulnerability and our experiments to all three ISPs. At the time of writing, the fixes had not yet been deployed.

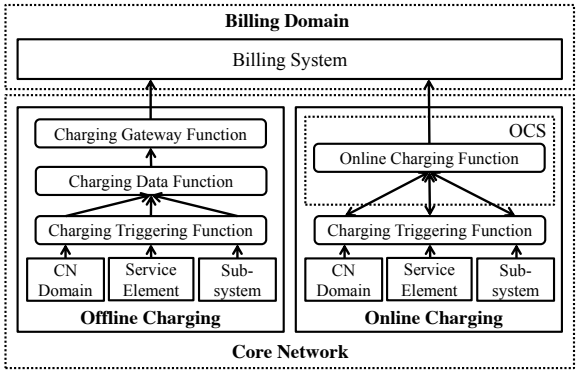


Fig. 3. Architecture of offline/online charging

the original IP packet, which makes IP packets the accounting units. (2) The GTP-U packet is transferred to the Gateway GPRS Support Node (GGSN) or Packet data network Gateway (P-GW), where it is converted back to the original Packet Data Protocol (PDP) format, in this case IP. (3) The GGSN records the packet received event for future billing and forwards the packet to the external data network where the target server is located. This process is done in reverse for the downlink case.

B. Cellular Data Accounting Process

Data accounting in cellular networks is done by the ISP by using Charging Data Record (CDR). The CDR is a formatted collection of information necessary for billing such as the user identity, the session, the network elements, and services used to support a subscriber session. Before the CDR creation, the user first sends an Activate PDP Context Request message to the SGSN/S-GW to initiate the PDP context setup process [20]. The PDP context activation then triggers the serving nodes (SGSN, GGSN, S-GW, P-GW) to create their own CDRs (S-CDR, G-CDR, SGW-CDR, PGW-CDR) with the UE's unique Charging ID (C-ID), and begins collecting the charging information. The SGSN/S-GW collects information related to the radio network usage while the GGSN/P-GW collects information on the external data network usage. The standard charging information collected by the serving nodes are the radio interface, usage duration, usage of the general packet-switched domain resources, source/destination IP addresses, usage of the external data networks, and the location of the UE.

Figure 3 shows the architecture of the offline and online cellular data charging mechanisms in the PS domain [21]. In the case of offline charging, the charging information for network resource usage is collected concurrently with the resource itself. During the charging process, a Charging Trigger Function (CTF) first generates a charging event, an activity utilizing network resources, and forwards it to a Charging Data Function (CDF). The CDF then constructs a CDR and transfers it to the Charging Gateway Function (CGF), which acts as a gateway between the CN and Billing Domain (BD) outside the CN. Finally, the CGF creates a CDR file and forwards it to a Billing System (BS) inside the BD where the billing actually takes place. Online charging is executed the same way except that the authorization for the network resource usage must be obtained by the Online Charging System (OCS) beforehand. The resource usage authorization may be limited by credits (i.e., duration or the data volume), resulting in different charging information compared to offline charging. During online charging, a CTF generates a charging event and forwards it to a OCS's Online Charging Function (OCF) in order to obtain authorization for the resources. The serving nodes deduct the resource usage until the available credit becomes zero. For byte-level accounting per user, most cellular ISPs account for entire IP packet sizes while their policies differ as to whether they include retransmitted TCP packets or not.

C. Possible Causes of TCP Retransmission

TCP guarantees reliable data transfer by retransmitting a packet when it suspects a packet loss [2], [4]. We analyze two major reasons for TCP retransmission in cellular networks: (1) packet losses in the path, and (2) bufferbloating at the base stations resulting in a large packet loss during a handoff.

1) Packet Drop Causes Retransmission: A packet drop may occur at three locations for cellular networks: (1) at a wired link between the server and ISP, (2) at a wired link between the GSNs/GWs within the ISP, and (3) at a wireless link between the client and RAN. Like the wired Internet, a packet drop for the first two cases is detected by the transport layer and is retransmitted after a node receives 3 duplicate ACKs or experiences an RTO. However, when a packet is dropped between a client and a base station in RAN, the link layer is responsible for retransmitting the packet until the receiving host acknowledges the packet [22], [23]. In other words, if a packet is dropped a number of times at the link layer but is eventually sent, TCP would not detect any packet losses but simply experience an increased Round-Trip Time (RTT). Only when a TCP timeout occurs does the TCP layer retransmit a packet to the receiving host.

2) Bufferbloating Causes Packet Loss at Handoff: A base station in a RAN typically has a large buffer in order to smooth out bursty data traffic for time-varying channels [24], [25]. A large buffer allows a base station to deal with incoming wired traffic rates above the wireless link capacity, but resulting in long latencies due to overbuffering as the base station tries to recover from packet drops. With an oversized buffer, a large chunk of packets may be lost when a inter-system handoff occurs such as between 3G and 2G, 3G and 4G systems [9]. This problem stems from the lack of support for packet handover between the RANs of different cellular networks. As a result, the original sender is required to retransmit every packet that is lost. If the base station maintains a large buffer, the packet loss rate would further increase, resulting in a large number of retransmissions.

III. TCP Retransmission Accounting Policy

To better understand the commonly used accounting policies for TCP retransmissions, we measured selected cellular providers in countries with varied cellular infrastructures. We used a *retransmission rate* metric defined as *(Retransmitted Bytes / Total Bytes in a Flow)*.

TABLE I. Accounting policies for TCP retransmission of 12 cellular ISPs in 6 countries

ISPs (Country)	Policy
AT&T, Verizon, T-Mobile, Sprint (U.S.)	blind
Telefonica (Spain)	blind
O2 (Germany)	blind
T-Mobile (England)	blind
China Unicom, CMCC (China)	blind
SKT, KT, LGU+ (South Korea)	selective

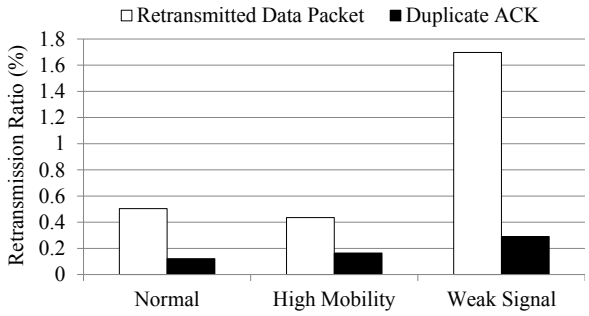


Fig. 4. Average retransmission rates in various environments

A. Accounting Policy for TCP Retransmission

We selected 12 ISPs in 6 countries (see Table I) over which we intentionally forced TCP retransmission packets and compared the transmitted volume with the accounted volume. Our test setup includes a test server on Amazon EC2 [26] and the following mobile clients depending on the availability for each ISP: Samsung Galaxy S3 (Android 4.1.2), LG Optimus Elite (Android 2.3.7), Samsung Illusion (Android 2.3.6), Apple iPhone 4 (iOS 5.1.1 - 9B206) and Apple iPad 2 (iOS 5.1.1 - 9B206).

1) Policy Measurement: We built a customized HTTP server sitting on top of a modified TCP layer that intentionally retransmits TCP packets without waiting for timeouts and ignoring ACKs. The server opens a standard TCP listening socket and a *raw* socket, and binds both sockets on the server IP and the listening port. It accepts the client connection via a listening socket and receives the HTTP request using the accept socket. It then uses a *raw* socket to inject apparent retransmitted packets (which may contain different payloads) in the on-going TCP session. On the client side, we used *wget* [27] to fetch a file from our server, and *tcpdump* [28] or *pirni* [29] to capture all packets in the connection. We then compared the captured packet volume with the accounted value by the ISP.

We investigate the retransmission accounting policy using the following two experiments. We first retransmit the same packet verbatim n times and measure the ISP accounted volume. We then run the same experiment, but we substitute the payload of the retransmission packets and again measure the ISP accounted volume. We ensure that the sequence number of the retransmission packets are within the last received ACK to prevent any middlebox from dropping the packets in transit.

2) Results: Table I shows the results of our experiments. Cellular ISPs in the U.S., China, and three European countries (i.e., all countries we investigated except South Korea) adopt a "blind" accounting policy that accounts for every IP packet regardless of TCP retransmission. In contrast, all three ISPs in South Korea enforce a "selective" accounting policy that does not charge for retransmission packets. Interestingly, we find that all of these ISPs are vulnerable to either "usage-inflation" or "free-riding" attacks, which we will explore further in Section IV.

The "Blind" accounting policy is understandable from an ISP's point of view as it is simple to implement and every packet consumes resources on the wireless cellular network. The rationale behind this policy is that the network should be responsible only for the IP layer functionalities while retransmission in the transport layer is regarded as an end-node issue. However, from a client's perspective, this could be unfair as the user might have to pay for packets that were discarded before reaching the application layer. That is, the subscribers have to pay for any packet that passes through the billing system regardless of whether it reaches the application on the client device or whether the same payload has been delivered multiple times and discarded by a lower layer in the device's communication stack.

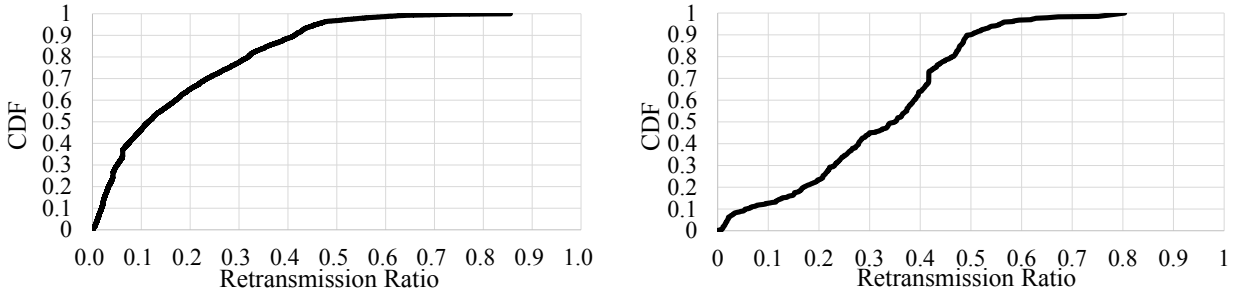
The "Selective" accounting policy attempts to be more fair to the subscribers. The underlying rationale is to hold the cellular ISPs responsible for packet retransmission since packet loss or delay is often caused by a lack of provisioning or operational errors. However, the implementation of selective accounting policy is complex, requiring flow state monitoring of about 10^5 to 10^6 concurrent connections in real networks. We find that one Korean ISP was actually identifying retransmission based on the sequence number (e.g., if a packet with smaller sequence number arrives out-of-order after supposedly the next packet, it is not accounted regardless of whether it is a retransmission or not).

B. Retransmission Rate Measurement

Given the accounting policies for TCP retransmission, we investigate the typical fraction of cellular traffic that can be attributed to retransmissions. To this end, we measure the TCP retransmission rates in devices carried by 11 volunteers, all graduate students at KAIST. The measurement period was 38 days (March 22nd - April 29th, 2013). During this period, we gathered 151,469 flows amounting to 3.62 GB of data in volume. All three ISPs in South Korea were used for the measurements. Although the sample is small and may not be representative of the cellular users population, we believe that the results give a first approximation on retransmission rates in the real world.

1) Retransmission Analysis: We implemented a simple packet analyzer that processes captured TCP packets in complete flows (i.e., 3-way handshake and teardown must be visible). The analyzer produces per-flow logs such as the total number of transferred packets and bytes, the retransmission rates of data packets and duplicate ACKs. We divide the flows by the application-level service to see if there is any difference in retransmission rates.

Average users do not experience retransmissions: Overall, the average retransmission rates are reasonable (0.4-1.7%)



(a) Daejeon in South Korea

(b) Princeton, NJ in the U.S.

Fig. 5. Cumulative distribution of the retransmission rates of the flows that experience any packet retransmission in South Korea and in the U.S. (The fraction of the flows with non-zero retransmissions is 10.7% in South Korea and 45.3% in the U.S.)

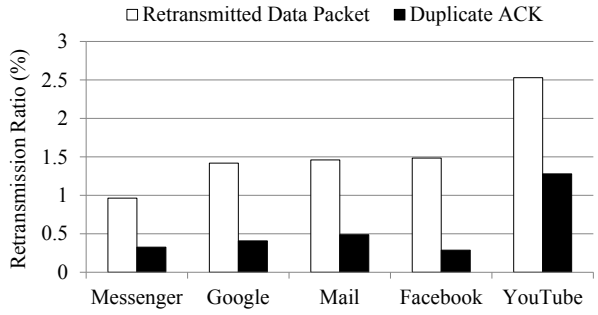


Fig. 6. Average retransmission rates per service. Messenger: KakaoTalk messenger [30], Google: Google Web search, Mail: KAIST main mail server, Facebook: Facebook mobile app, YouTube: video streaming

as is shown in Figure 4. We first confirm that all retransmission packets are legitimate, having the identical payload as the original packets. Then, we divide the measurement environment into three groups based on factors that can affect packet drops and retransmission rates. For stationary users, we tag the flows as "normal" if the signal strength is strong (4-5 signal bars) or "weak signal" otherwise. For moving users (e.g., in a travelling car or in a high-speed train), we tag the flows as "high mobility".

In our sample, the average retransmission rates for the "high mobility" group are similar to those in the "normal" group, both for retransmitted data and duplicate ACK packets (0.5%, 0.12% vs. 0.44%, 0.16%). This implies that the handovers work well without a high packet loss rate. For the "weak signal" group, the retransmission rates were slightly higher (1.7%, 0.29%). We suspect that links with weak signals incur more packet drops due to timeouts or increased errors in the wireless environments.

When the data collected was organized by application layer services, the retransmission rates appeared roughly similar. Figure 6 shows the comparison of five of the most popular services in our volunteer sample. The notable exception was YouTube showing slightly higher retransmission rates than others. This is likely due to the application flow control techniques used by YouTube [31]. We also suspect that larger contents often cause short-term congestion, and increase the retransmission rates. Additionally, many YouTube videos may be cached at CDN outside South Korea, increasing the likelihood of out-of-order delivery which in turn would trigger the TCP fast retransmit mechanism.

Some flows experience high retransmission rates: While the majority of flows have low retransmission rates, our data shows that some flows experience high retransmission rates. Figure 5(a) shows the cumulative distribution of retransmission rates for any flows that had at least one retransmitted packet. Over half of them show more than 10% retransmission rates, and the worst 10% has retransmission rates between 40 to 85%. Those measurements are in line with our previous measurements in the 3G cellular backhaul link in which we find some flows have as large as 93% retransmission rates [32].

To see if a similar situation is observed in other countries, we measure the retransmission rates with one of the major ISPs in Princeton, NJ. The overall retransmission rates are slightly higher than in South Korea but are reasonable, ranging from 2.2 to 3.2% on average, although some flows exhibit high retransmission rates. Figure 5(b) shows that over half of the flows show more than 35%, and the worst 10% shows over 49% and up to 80% retransmission rates. Given that we captured the packets on the end client device, the actual retransmission rates could be higher if some of retransmitted packet were dropped in transit.

IV. TCP Retransmission Attacks

We implement the "usage-inflation" and "free-riding" attacks and demonstrate that one can easily abuse the cellular accounting systems of current ISPs with malicious TCP retransmissions. We ran the "usage-inflation" attack on four ISPs in the U.S. (AT&T, Verizon, Sprint, and T-Mobile) that blindly account for retransmission packets. We ran the "free-riding" attack on three ISPs in South Korea (SKT, KT, LGU+) that do not account for TCP packet retransmission. In our experiments, the "usage-inflation" attack was attempted only on our mobile clients and we paid for the resulting inflated amounts. We have also informed the South Korean ISPs of the "free-riding" attack prior to our experiments. The experiments are carried out in Princeton, NJ in the U.S., and in Daejeon in South Korea.

A. Usage-Inflation Attack

The "usage-inflation" attack arbitrarily inflates the cellular data usage of a target subscriber by intentionally retransmitting packets in the flow even without actual packet losses. Figure 7 shows one attack scenario. The attacker first sends a phishing

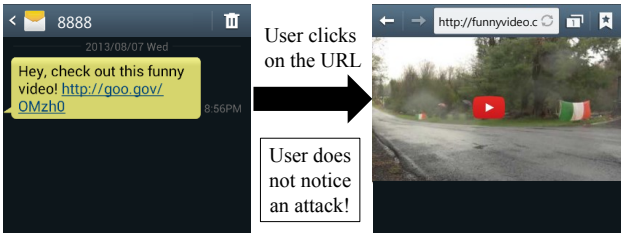


Fig. 7. "Usage-inflation" attack. A phishing SMS message is sent to a target client. When the user clicks on the link, she is redirected to an innocent-looking webpage while the attack is launched in the background.

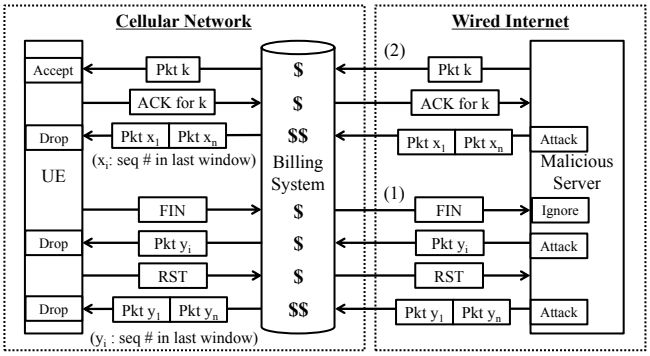


Fig. 8. "Usage-inflation" attack process. 1) Retransmission attack after the client-side connection closure. 2) Retransmission attack along with normal data transfers.

SMS message to a target client with the URL that leads to a malicious site. When a user clicks on the link, the browser is redirected to that malicious Web server that inflates the usage by spurious packet retransmission. At the application layer, the server transfers the requested content to the client in a normal TCP connection, so the subscriber does not suspect any sign of attack. However, the underlying modified TCP layer on the server behaves as if it did not receive any ACKs from the client or as if its RTO fired prematurely and injects retransmitted packets in the background. This attack does not require compromising the client or any intermediate hops. As long as the user is redirected to a malicious server (via 3rd party advertisements, phishing emails or SMS messages), the attacker can inject any number of retransmission packets, which does not violate the TCP semantics.

Figure 8 shows two ways to implement this attack. First, the attacker can retransmit the packets after the client finishes downloading the content. That is, the server pretends it did not receive the client-side FIN/RST as well as the ACKs for the last batch of packets and retransmits the packets that belong to the last send window. This approach allows an attacker to greatly overcharge the usage in a short time by utilizing the full bandwidth between the server and the client. However, a smart accounting system may prevent the attack if it detects a high retransmission rate after a FIN/RST. A more sophisticated attack is to embed the retransmission packets in the stream of normal packets. Instead of blindly retransmitting the same packet multiple times, the attacker can carefully pick a random packet in its send window. To prevent a noticeable slowdown in the download rate, the attacker can control the goodput

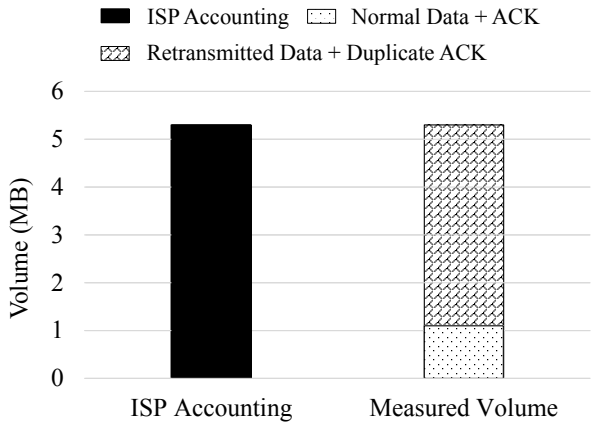


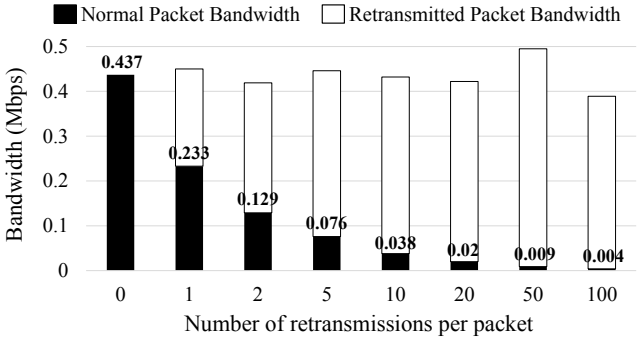
Fig. 9. Accounted volume vs. measured volume for "usage-inflation" after FIN for all U.S. cellular ISPs

of interactive contents (e.g., audio stream typically requires 500 Kbps while video streaming requires 2 Mbps [33]) while injecting the retransmission packets in the background. As long as the streaming content does not stop in the middle, the user may not notice an ongoing attack.

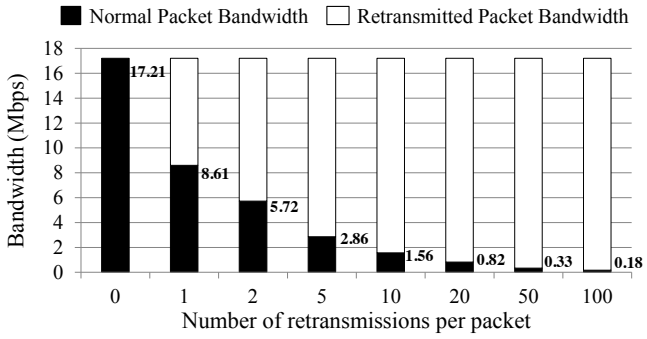
Retransmit after FIN: We instantiate this attack by downloading a 1 MB file via `wget` from a server that we control. Our server retransmits four copies of each packet after the FIN (a usage blowup factor of four). We conduct the experiments with all four U.S. ISPs, and compare the captured volume on our mobile device with the accounted value by the cellular ISPs. We find that the attack is successful with all four ISPs, and we show the result of one of the attacks in Figure 9.

To determine the limits of usage inflation after a client-side FIN, we continue to send retransmission traffic from the server and note the client side behavior. Once the client closes the connection by sending a FIN, it continues to reply to all incoming retransmission packets with ACKs for 2-3 minutes. After that period, it switches to sending RST packets in an attempt to get the remote server to stop the flow of TCP packets for this connection. We find that some ISPs drop all retransmission packets after the RST while others deliver the packets to the client for as far as we could observe. In fact, we find that one ISP allows retransmitted packet transfers up to 4 hours after we stopped the experiment following the FIN packet from the client. We also find that all US ISPs do not limit the data transfer speed after the FIN, allowing us to inflate the usage to 1.14 GB in just 9 minutes with one ISP. Table II shows the packet drop policy after a RST with 7 ISPs in the U.S. and in South Korea. The ISPs noted on the table are anonymized and in a random order.

Retransmit during Normal Transfer: Large client downloads that lasts for long time periods are vulnerable to malicious packet retransmission during normal data transfer. For example, if a user watches a streaming video that spans tens of minutes, the attacker can inflate the usage in the background while still meeting the bandwidth requirement for the content. Compared with the retransmit-after-FIN attack, this attack is harder to defend since the previous attack could be mitigated at the ISP by blocking any incoming packets after a client-side RST.



(a) Results for one ISP in U.S.



(b) Results for one ISP in South Korea

Fig. 10. Effective bandwidths over various usage blowup factors with an ISP in the U.S. and in South Korea

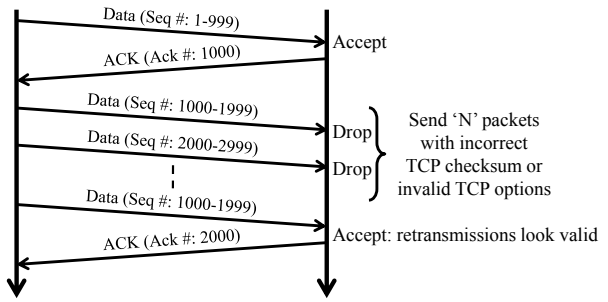


Fig. 11. "Usage-inflation" attack with incorrect TCP checksums or invalid fields in TCP headers/options

We simulated the retransmit-during-normal-transfer attack by having the client download a 10 MB file and saturate the available bandwidth with various *usage blowup factors*, n (e.g., n = 1, 2, 5, 10, 20, 50, 100). We confirmed that all ISPs show a usage inflation as expected, shown in Figure 9. Our attack didn't require much sophistication (such as avoiding too many retransmissions of the same packet) on current ISPs since they blindly accounted for all retransmission packets into the customer bill.

Next, we measure the available bandwidth under various usage blowup factors. The measurement is important to keep the goodput above the minimum bandwidth requirement of the client-side application during our retransmission packet injection attack happening in the background. Figure 10(a) shows the goodput over various usage blowup factors with one ISP in the U.S. As expected, the goodput degradation is inversely proportional to the usage blowup factor. While the maximum available bandwidth is not large in the area, we believe that the attacker can squeeze in more spurious packets as the wireless bandwidth in the region improves. Figure 10(b) shows the available bandwidth for the same experiment with one ISP in South Korea.

Figure 11 shows a more sophisticated attack scenario. To make the retransmission look legitimate, the attacker manipulates the TCP packets such that the server sends the packets with incorrect TCP checksums or invalid values in TCP headers or options (e.g., an invalid URP pointer or invalid Maximum Segment Size (MSS) option [4]). Then, the client would silently drop the received packets without responding with ACKs. Subsequent retransmission packets from the server would look legitimate to the accounting system since it has not seen the ACKs. We have tested these scenarios with all 7 cellular ISPs in the U.S. and South Korea and find that all U.S. ISPs drop the TCP packets with wrong values while some ISPs in South Korea forward them to the client. Interestingly, this implies that some ISPs verify the TCP header/option fields in the accounting middlebox. Especially, TCP checksum verification would require scanning the payload of the packet, so DPI on every TCP packet is already in place with some ISPs [3]. However, we also note that if an attacker decides to bloat the TCP headers with valid options and checksums (e.g., fill up TCP option with NOOP), DPI would still not detect the inflation. Table II summarizes the packet handling policies of the 7 ISPs anonymized and in a random order.

TABLE II. Packet handling policies in U.S. (1-4) and South Korean ISPs (5-7)

ISP	Pkts after RST	Pkts with incorrect checksums	Pkts with invalid options
1	block	block	block
2	block	block	block
3	forward	block	block
4	forward	block	block
5	forward	block	block
6	block	forward	forward
7	forward	forward	forward

Difficulty of Detecting "Usage-Inflation" Attacks: It may appear that the cellular accounting system can detect the "usage-inflation" attack by catching the anomalous flows with abnormally high retransmission rates. However, detecting the attack by a static retransmission rate threshold could lead to false positives since those exist in legitimate flows as we have previously observed in Section III. If a smart attacker controls the level of usage inflation such that the retransmission rates look similar to those of poorly-provisioned regions, detecting the attack could be challenging.

For accurate detection, the cellular accounting system has to monitor the TCP sender behavior. For example, it can observe the sender's windows after a retransmission to ensure

[3]TCP checksum verification is often implemented in hardware.

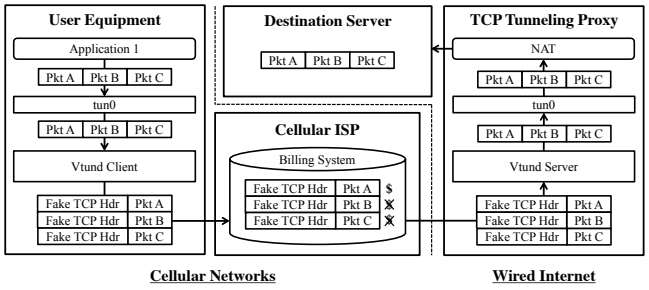


Fig. 12. "Free-riding" attack process (uplink case)

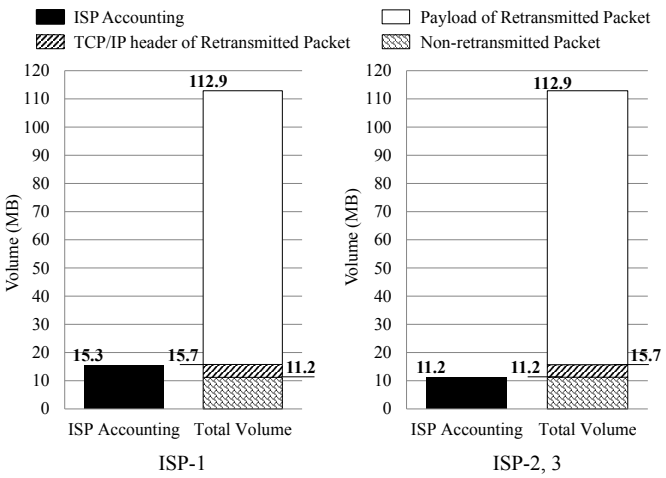


Fig. 13. "Free-riding" attack results

that it is reduced as expected. However, this mechanism is analogous to detecting selfish senders that do not follow the standard TCP congestion control algorithm. Detecting selfish senders in a middlebox is non-trivial since it is hard to reliably distinguish if a TCP flow action is caused by the most-recently forwarded packet or the one in the distant past [10]–[12].

One possible workaround is to retain some TCP state in the core network. In this instance, the accounting system can manage the client-side flow state such that it does not account for any data packet sent by the server that has been already ACK'ed by the client. That is, if the accounting system observes the client-side ACK, for billing purposes it could ignore any retransmission packet whose sequence number is smaller than the ACK. The accounting system still needs to forward the retransmission packet to the client, since the previous ACK could have been lost in the path between the accounting system and the server. This solution can be brittle since a naive implementation that blindly allows those retransmissions by only observing the header could miss tunneling traffic in the payload as discussed in the following section.

Another workaround is to use a TCP proxy that relays every TCP connection between the client and the server. The accounting system can more accurately measure the data exchanged between the TCP proxy and the client. This might be viewed as extension of Performance Enhancing Proxies (PEPs) already deployed in some cellular ISPs for Web traffic [6]. The main drawbacks in splitting every TCP connection is the cost given increasing mobile traffic, in addition to the TCP proxy itself becoming the target of other attacks.

B. Free-riding Attack

An adversary can launch a "free-riding" attack against cellular ISPs that take out retransmission packets from the users' billing totals. The attack avoids the accounting mechanism by tunneling another TCP connection inside of a fake TCP header that masquerades as a retransmission. Figure 12 illustrates our implementation. The "free-riding" attack requires a collaborating TCP tunneling proxy that relays the tunneled packets and real traffic between the client and the server. For upstream traffic, packets from the client are tunneled to the proxy using fake TCP headers and de-tunneled at the proxy and relayed to the destination server. For downstream traffic, the packets from the server arrive at the proxy which tunnels them to the client, and the client de-tunnels and passes them to the application. In this architecture, the accounting system in the cellular core network will only see the connections between the client and the proxy with large numbers of retransmissions, which will not be counted in the final bill.

We implement the TCP tunnel using the Universal TUN/TAP driver [34]. On the client side, we run a modified VTun daemon (vtund) that captures all the packets from applications and inserts new TCP and IP headers destined to the tunneling proxy. For tunneled packets, we set the Maximum Transmission Unit (MTU) to "original MTU size - TCP/IP header size" to prevent IP-level fragmentation and reassembly between the client and the proxy. On initialization, vtund creates a TCP connection to the proxy located in its configuration file, and multiplexes all TCP packets from the applications on the client device over the client-to-proxy connection. We allow the client to configure the *retransmission factor*, which represents the number of retransmission packets it would send per packet on average. The attacker can leverage this factor to decide the level of free riding. To evade the tunnel header detection by the accounting system, vtund encrypts the payload of all tunneled packets using RC4 [35]. Optionally, it applies packet compression using the LZO algorithm [36] which is popular for real-time packet compression/decompression.

The tunneling proxy detunnels the packets from the client and relays the original packets to the real destination. After decrypting (and decompressing) the payload and extracting the original packet, the proxy forwards it to a Network Address Translation (NAT) table. The NAT is responsible for translating the client-side IP address and port number to those of the proxy and vice versa. The downstream scenario works in the same way. The proxy tunnels the packets from the server to the client. Encryption and compression is applied to those packets as configured in the proxy setting.

This attack is simple to implement and easy to use. The vtund code modification consists of about 600 lines of code ($\sim$300 lines for tunneling and $\sim$300 lines for encryption and compression). In order to use the virtual network interface (e.g., *tun0*), however, one needs to root or jailbreak the client device, but no further system-level modification is needed since vtund runs in user space.

Free-riding Attacks in Practice: We test the attack on three South Korean ISPs that do not account for retransmis-

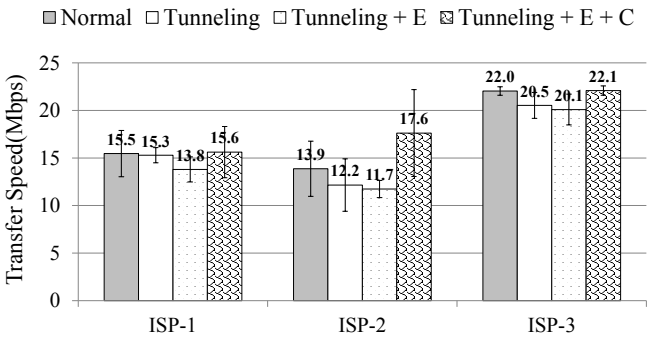


Fig. 14. Comparison of download throughputs with SpeedTest. "Normal" implies innocent usage while "tunneling" implies the "free-riding" attack. E=encryption, C=compression.

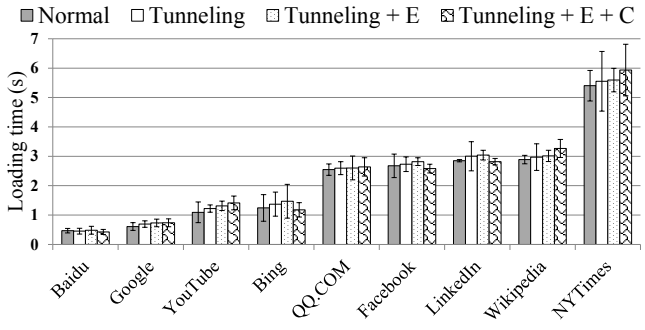


Fig. 15. Front page loading time with popular Web sites. Normal implies innocent usage while "tunneling" implies the "free-riding" attack. E=encryption, C=compression.

sions. We run the tunneling proxy on a machine with an Intel Core i7 860 CPU (2.80 GHz, quad core), 4 GB physical memory, and a 1 Gbps network interface card (NIC), and used a Samsung Galaxy S3 as the client. The proxy runs on Ubuntu 12.04 and the client runs on Android 4.2.2. The experiments were executed by downloading a 100 MB file via wget. We have exercised the tests with various retransmission factors (up to 100), and confirmed that all of the attacks work successfully with these ISPs. Figure 13 compares actual transferred content size vs. accounted volume by ISPs, using a retransmission factor of 10. While we see a slight difference in the accounted volumes depending on the accounting policies regarding retransmission packet headers (e.g., one ISP accounts for TCP/IP headers of retransmission packets while the others do not), we find that an attacker can use between 7.4 and 10.1 times more cellular data than accounted for by the ISP.

We next measure the effective bandwidth used by the tunneling proxy. First, we use the SpeedTest app [37] to measure the download throughputs from a server located in Seoul, South Korea. We repeated the tests five times and calculated the average numbers. Figure 14 compares the results in the presence or absence of tunneling for all three ISPs. We see that the download throughputs with tunneling are between 15.6 to 22.1 Mbps when we apply both encryption and compression. These are very similar to those without tunneling, which implies that the tunneling overhead can be hidden by compression. The tunneling throughputs without compression show 11.7 to 20.1 Mbps, which is 9% to 16% lower than the normal throughputs due to the additional overhead of tunneling and encryption. The throughputs appear adequate for practical use.

To estimate how practical this attack can be, we measure the front page loading time with nine popular Web sites for smartphones, including Facebook, YouTube, Bing, NYTimes, etc. We use the Chrome Web browser in combination with WebWait [38] and repeat the experiments 10 times. To reduce errors, we flush the browser cache before loading a page. Figure 15 shows the average loading time with one cellular ISP. Overall, tunneling shows similar loading times. Tunneling traffic with encryption but without compression, we observed a 9.3% slow down in average loading time. With compression, we observed a modest speedup compared to normal browsing. The latency increase is mainly due to the overhead from extra tunneling headers, encryption, compression, and at least an extra hop on the network path to include the proxy. However, we believe the free cellular data usage is still an attractive option despite slight performance degradation.

Difficulty of Detecting "Free-Riding" Attacks: The accounting system might attempt to detect the "free-riding" attack in two ways. First, it could try to detect spurious retransmission. However, this approach has the similar difficulty as in the "usage-inflation" attack. As long as the attacker sets the retransmission factor to be low enough and adds enough randomness in retransmission, it would be non-trivial to reliably detect the attack. In addition, since the proxy and the client collude with each other, they can make the retransmission packets completely legitimate by intermittently exchanging ACK packets. Furthermore, the client can spread the TCP packets over multiple colluding proxies, which would make the attacker harder to detect by an accounting system.

Second, the accounting system may attempt to compare the payloads from the original and retransmission packets. One efficient approach might be to utilize the TCP checksum. That is, the accounting system stores the TCP checksum of the original packet and sees if the retransmission packet with the same starting sequence number carries a matching TCP checksum. Non-matching packets can be safely discarded. This policy looks reasonable since it can avoid expensive byte-by-byte comparison of the original and retransmission packets from full DPI. However, we find two problems here. First, in order to verify the checksum of the retransmission packets, the accounting system has to traverse all payload bytes to compute the checksum. This could be expensive on high-speed cellular core networks. Second, the TCP specification allows the retransmission payloads to be of a different size from the original packets [4]. If the attacker indeed uses a different payload size for the retransmission packet, the accounting system cannot rely on the checksum of the original packet. Later, we propose a light-weight DPI that reliably detects the attack on commodity hardware.

V. Defending against Free-Riding Attacks

We consider defense strategies against the TCP retransmission attacks discussed in the previous section. If an ISP adopts a "blind" accounting policy that charges for retransmission

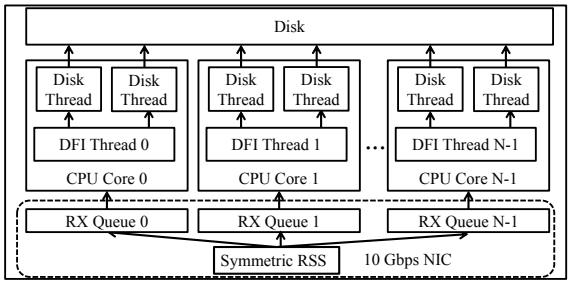
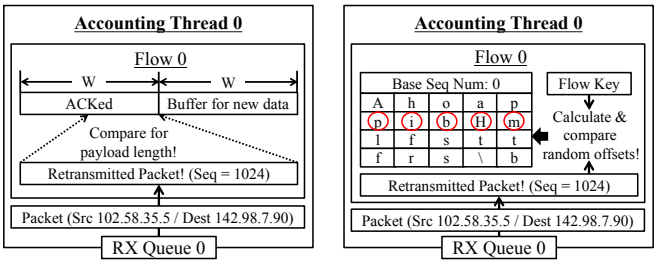


Fig. 16. Overall architecture of Monbot. N is the number of cores in a machine.



(a) Deterministic DPI (b) Probabilistic DPI

Fig. 17. Attack detection with retransmission packets

packets, it has to protect its subscribers from spurious retransmission. Unfortunately, we do not know of any reliable way to detect such attacks in a middlebox without false positives. If the attackers are smart enough to simulate the behavior of poorly-provisioned areas, it would be non-trivial to detect the attack. Instead, we focus on detecting "free-riding" attacks in the ISPs with a "selective" accounting policy that ignores legitimate retransmission. We implement an accounting system called Abacus that runs DPI on the retransmission packets to detect tunneling attacks. Abacus extends Monbot [5], a highly-scalable flow monitoring system on commodity hardware, to drastically reduce the flow buffer requirement by probabilistically verifying the payload of retransmission packets. We first provide some background on Monbot, and show how Abacus implements deterministic and probabilistic DPI that can handle on the order of 10^5 concurrent flows in high-speed cellular core networks.

A. Background of Monbot

Figure 16 shows the overall architecture of Monbot. The key design principle is parallel and batch processing. For high-speed packet I/O, Monbot uses the PacketShader I/O Engine (PSIO) [39] that bypasses the heavyweight kernel networking stack and delivers a batch of incoming packets from NICs into the user-level process at speeds between 10 Gbps and 100 Gbps. When packets arrive at the system, they are evenly distributed to multiple CPU cores using Symmetric Receive-Side Scaling (S-RSS) [5]. S-RSS ensures to map the packets in the same TCP connection to the same CPU core. This way, a thread affinitized to one CPU core can process all packets in a TCP connection without sharing the flow management state with other threads. This eliminates lock contention and improves the scalability with multiple CPU cores. Flow management is implemented on each Deep Flow Inspection (DFI) thread independently of each other. Depending on the available memory, Monbot allows 400K to 1 million concurrent flows on a 10 Gbps network.

Monbot has been installed on a cellular core network of one of the largest commercial ISPs in South Korea. It has monitored one of the ISP's 10 Gbps links for seven days without a single packet drop. It has analyzed 370 TBs of packet data and 8.3 billion TCP connections, and produced a SHA1 hash per each 4KB content chunk in all TCP connections. For the measurement period, we have seen as many as 270K concurrent flows (including idle flows) and 1.1 million TCP connections per minute [5].

Abacus extends Monbot to accurately account for a large number of concurrent flows while reliably detecting any "free-riding" attacks. It reuses the packet I/O and flow management modules of Monbot but disables other features not needed for cellular accounting such as SHA1 hash calculation and TCP/HTTP-level traffic analysis. Abacus produces per-flow accounting logs such as numbers of total accountable bytes, transferred packets, innocent retransmission packets (including duplicate ACKs) as well as retransmission packets suspected to be malicious. For this, we have modified 1,880 lines of the original Monbot code. To catch the "free-riding" attack, Abacus supports DPI in two different modes, which we will explain in the following sections.

B. Deterministic DPI

Deterministic DPI (d-DPI) buffers the original TCP segments and verifies that retransmission and original packets have identical contents. It conducts byte-by-byte comparisons of the original and retransmitted payloads, and flags an attack if there is any disparity. One challenge in d-DPI is to determine the buffering amount of a flow. Theoretically, one needs to buffer (RTT x bandwidth) amount of flow data, but since per-flow RTT and effective bandwidth change on a short time scale, we use the receive window size (e.g., advertisement window size) in each direction of a TCP connection. Since TCP flow control dictates that the sender's window should not be larger than the receive window size, this value should give an upper bound of the send window on one end.

Implementation: d-DPI is straightforward to implement. Abacus creates a new flow entry when it sees a 3-way handshake of a TCP connection. Each flow entry has two flow buffers whose sizes simulate those of the send windows of the client and the server. We calculate the amount of flow data that Abacus holds as follows. Let the largest sequence number the Abacus has seen from one end as S. We estimate the maximum send window size as the receive window size, W, advertised by the receiver. Then, the system needs to buffer any sequence numbers $\geq$ (S - W) assuming no out-of-order packet delivery. However, W could change on every ACK and packets can be delivered out of order in practice. So, we decide to buffer any sequence numbers $\geq$ (S - 2 x W) where W is the maximum receive window size advertised by the other end. As S advances, we slide the flow buffer window to monitor the right window range.

We optimize the buffer usage by lazy memory allocation. Instead of allocating the entire flow buffer to the maximum

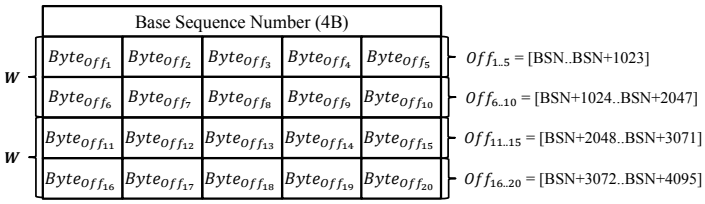


Fig. 18. Flow Table with Sample Entries for p-DPI (n = 5)

receive window size, we allocate a small buffer (e.g., 4KB) initially and switch to a larger buffer (e.g., as much as 16 MB) as we need to buffer more packets. To implement this, Abacus benefits from efficient user-level memory allocation of Monbot that reduces the overhead from frequent memory re-allocation. More information can be found in [5].

C. Probabilistic DPI

While d-DPI guarantees to detect all possible "free-riding" attacks in malicious retransmission, it requires a large amount of memory and high memory bandwidth in high-speed networks. We overcome these weaknesses with probabilistic DPI (p-DPI). The basic idea is to store the packet data by sampling and see if the retransmission packets have the identical values for the sampled data. For example, if we sample 5 bytes per each 1000-byte packet, we can reduce the buffer memory requirement by a factor of 200. In this way, we can significantly reduce the memory size and memory bandwidth.

Implementation: The implementation of p-DPI (p-DPI) is similar to that of d-DPI except that p-DPI buffers sampled flow data. p-DPI samples n bytes per each 1024-byte flow data at random. For this, we allocate a flow table per each flow direction (e.g., upstream and downstream) that consists of a set of *sample entries* (Figure 18). Each sample entry has a 4-byte base sequence number (bsn) and n-byte sampled data. Each sampled byte on the entry is randomly chosen from the sequence number space of [bsn, bsn + 1023]. The bsn for the first sample entry is set to the initial sequence number (isn) taken from the SYN packet (and SYN/ACK packet for the opposite direction), and it increments by 1024 bytes per each entry in the flow table. The window of the monitored flow data is tracked in the same way as in d-DPI.

To limit successful guessing of the sampled byte locations, we randomize those locations for each flow. The sampled byte locations in each entry are determined by running a hash function with (per-flow secret key, bsn for the entry) as input. The per-flow secret key is generated by HMAC_{secret_key}(nonce) at connection setup time where the nonce is a 8-byte random number generated per each flow and the secret key is the system-wide key known only to Abacus. The secret key is rotated once every day when the system load is low (e.g., early in the morning). Any hash function is fine as long as its output size is (10*n) bits or larger. For the current implementation, we use the Bernstein hash function [40] that produces a 64-bit output.

The sampled byte location is calculated as follows. Suppose the hash function produces K as the output. Then, we determine the offsets of the n bytes as $K_{[0..9]}$, $K_{[10..19]}$, ..., $K_{[10(n-1)..10n-1]}$ where $K_{[x..x+9]}$ represents a 10-bit number taking the x^{th} to $(x+9)^{th}$ bits from the least significant bit position. That is, we sample n bytes whose sequence numbers correspond to

$$(bsn + K_{[0..9]}), (bsn + K_{[10..19]}), ..., (bsn + K_{[10(n-1)..10n-1]})$$

In this way, we avoid storing the offsets or sequence numbers for each sampled byte location. Consequently, we trade the memory space and memory bandwidth for hash computation, which is a reasonable choice given the trend that the processor speed is much faster than memory bandwidth.

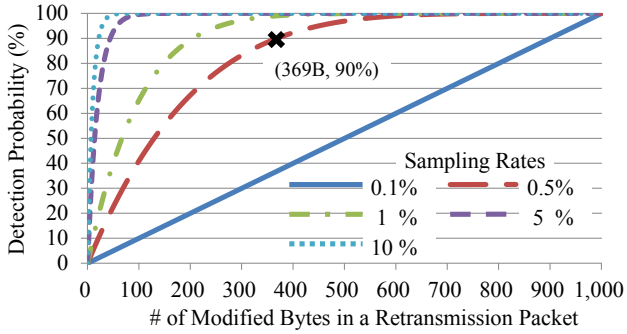


Fig. 19. "Free-riding" attack detection probability for various sampling rates (0.1%, 0.5%, 1%, 5%, 10%).

When a retransmission packet arrives at Abacus, it looks up a relevant flow entry with four tuples (e.g., source and destination IPs and port numbers) of the connection. If the flow entry exists, the system calculates the bsn for the sample entry that the packet belongs to. Then, it runs the hash function with the bsn and the flow key as input, and finds the sampled locations for content comparison. In case the sequence number space of the packet spans over more than one sample entry, it repeats the process for others as well. Given that a typical MTU size is 1500 bytes, the verification process per each retransmission packet ends in one or two rounds.

Choice of n: In choosing n, the administrator faces a tradeoff between memory space efficiency vs. attack detection accuracy. As we decrease n, we can reduce the required buffer memory but risk incurring false negatives, allowing the attacker to evade the detection. On the other hand, a large n can increase the detection accuracy, but the benefit of p-DPI decreases. Here, we probe the right size of n by calculating the detection probability. Assume Abacus samples y bytes from an original packet and the attacker employs x bytes of the retransmission packet for the "free-riding" attack while keeping the remaining bytes intact. For a packet size of 1000 bytes, the attack detection probability in the retransmission packet is

$$1 - \frac{{}_{(1000-y)}C_x}{{}_{1000}C_x} \quad (1)$$

where ${}_aC_b$ represents $\binom{a}{b}$. Figure 19 shows the detection probability of an attack packet as a function of x over different sampling rates. We see that even for a small n such as 5 (0.5% for a 1 KB packet), we can detect the attack with 90% or higher probability if the attack packet modifies 36.9% or larger fraction of the payload. Furthermore, if the attacker uses multiple attack packets in a flow, the detection probability gets

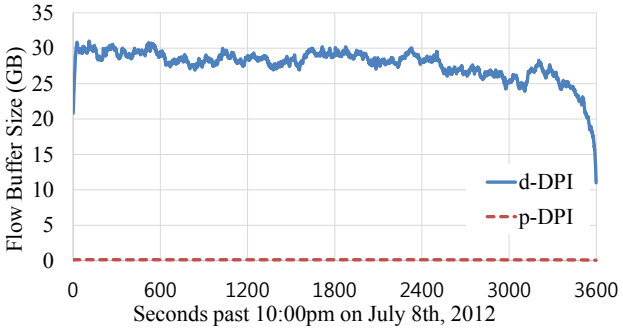


Fig. 20. Aggregate flow buffer sizes of d-DPI and p-DPI on a 10 Gbps cellular core link (10-11pm KST, 7/8/2012).

quickly close to 100%. This means that as long as we sample any positive number of bytes in each packet, we can detect all practical "free-riding" attacks with a high probability.

D. Flow Buffer Requirements

To estimate the aggregate flow buffer size needed for d-DPI and p-DPI in a real cellular core network, we collected the maximum receive window sizes of all TCP connections at a 10 Gbps 3G core link of one of the largest commercial ISPs in South Korea [5]. We measured the data for one hour in the peak time of a working day (10pm to 11pm on July 8th, 2012) and calculated the aggregate buffer size at each second. The number of concurrent TCP connections during the time ranges from 45,647 to 114,213 with an average of 103,183.

Figure 20 compares the buffer memory requirements of d-DPI and p-DPI. The sampling rate of p-DPI is set to 0.5% (n=5 per 1 KB). We see that the memory requirement of d-DPI ranges from 11 to 31 GB with the average of 27 GB while p-DPI requires 0.05 to 0.15 GB with the average of 0.13 GB. As expected, p-DPI reduces the memory requirement by a factor of 200, and it implies that it can scale to 1 million concurrent flows with as small memory as 2 GB.

VI. Evaluation

A critical requirement for an effective defense is scaling to large numbers of concurrent flows. We evaluate the performance of Abacus for detection the "free-riding" attacks in a realistic setting to see how well it can meet this requirement. To generate flows, we run a custom HTTP server paired with a traffic generator client, pulling data from the server. Both the server and client were running on machines with dual Intel Xeon E5-2690 CPUs (2.90 GHz, octacore), 64 GB of physical memory, and an Intel 10G NIC with 82599 chipsets. We tested Abacus in d-DPI mode on a machine with the same specification as the server, but tested the p-DPI mode on a desktop-level machine with an Intel i7-3770 CPU (3.40 GHz, quadcore) and 16 GB of physical memory with a 10G NIC. All machines ran on Debian 6.0.7 (Linux kernel ver:2.6.32) and were connected to a 10 Gbps Arista 7124 switch. Abacus monitors all the packets exchanged between the server and the client via port mirroring.

A. Microbenchmark

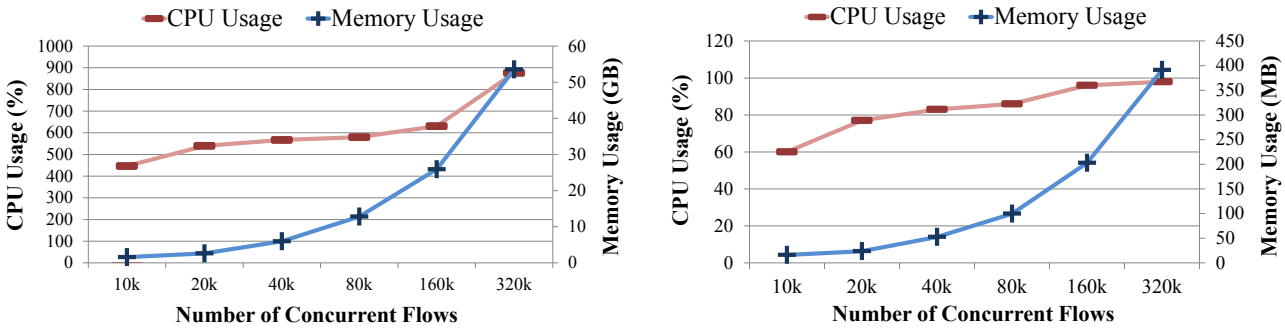
We measure the scalability of Abacus against a variable number of concurrent flows. On the client side, we run a custom traffic generator that repeatedly downloads a small file (12 KB) from the server over HTTP. The traffic generator creates a new connection for each file download. We monitored the behavior of Abacus as we increased the concurrent flows from 10K to 320K. The Linux TCP stack does not scale well on a multicore system with a large number of concurrent flows [41], so we used a user-level TCP stack on PSIO that we built for the server and the client. Further details of our TCP stack is beyond the scope of this paper. Our user-level TCP stack scales linearly to the number of CPU cores and saturates the 10 Gbps link even when the flow size is small.

Figure 21(a) and Figure 21(b) compare the CPU and memory consumption of Abacus over a varying number of concurrent flows. The accounting volume by Abacus is accurate: it produces the correct number of total transferred bytes except retransmission packets. The traffic between the server and the client fully utilizes the 10 Gbps link regardless of the concurrency level. We observe that d-DPI works well up to 160K concurrent flows but it starts to drop packets at 320K flows. The memory usage grows linearly with the number of flows, showing 25.9 GB at 160K flows and 53.6 GB at 320K flows. In contrast, p-DPI does not drop any packet even with 320K concurrent flows. The memory usage is 391.0 MB at 320K flows and 202.7 MB at 160K flows. The CPU consumption of d-DPI stays around 500% to 600% (where, 100% = 1 CPU core is fully utilized) for 160K flows and below but grows to 876% at 320K flows, presumably due to the bottleneck in the memory bandwidth. In contrast the CPU usage of p-DPI stays under 100% even at 320K flows, indicating a good scalability for Abacus in p-DPI mode for a large number of flows that saturate a 10 Gbps link, even on a low-powered desktop machine.

B. Real Traffic Simulation

We measure the performance of Abacus against the "free-riding" attacks in a realistic environment. We build a cellular traffic generator that replays the 3G cellular traffic logs measured in a commercial cellular ISP in South Korea [5]. Each entry in the logs consists of the SHA1 hash of the entire flow content and a list of 4 KB content chunk hashes along with a total size and start/end timestamps. The client issues an HTTP request with the content hash as the key and the server dynamically creates a response that consists of the chunks matching the content key and sends it to the client. The content of each chunk is filled with a repeated pattern of SHA1 chunk hashes. We originally built this for testing redundancy elimination (RE) caching proxies, and found it to be a good fit to simulate application layer content over 3G networks. The logs were extracted from the real 3G traffic at a daily peak hour (from 11pm to 12am on July 7th, 2012) in the commercial cellular ISP. The number of replayed flows was 61 million, totaling 2.79 TB in volume, and the average bandwidth during the time was 4.35 Gbps. To reduce the experiment time, we replayed the logs slightly faster so that the bandwidth stays between 5 to 8 Gbps.

During the log replay, we injected 100 flows that simulated "free-riding" attacks. Each flow randomly adds retransmission



(a) d-DPI (b) p-DPI

Fig. 21. Comparison of CPU and memory usages of Abacus in d-DPI and p-DPI modes

packets with different payload from the original packets. We find that d-DPI and p-DPI report all of the attack flows accurately.

VII. Related Works

A. Cellular Network Analysis

There have been a number of works that characterize the cellular network traffic. Woo et. al. characterize the 3G traffic at a 10 Gbps backhaul link of a large commercial ISP, and report that over 95% of the 3G traffic is based on TCP and up to 59% of the traffic is redundant on the content level [5]. They find that the most flows are small and short-lived with the number of concurrent flows as large as 270K. Huang et. al. report similar characteristics (e.g., the dominance of TCP, short flows, and small sizes) in an LTE network [6]. They find that the retransmission rates are low in general such that 38.1% of the flows have zero retransmission and note that most packet drops are masked by physical/MAC-layer retransmission. While we also observe low retransmission rates with average flows, we do find that a small number of flows suffer from high retransmission rates, which can be exploited by TCP retransmission attacks. They also reveal that PEPs intercept all TCP connections with server port 80 or 8080. These proxies terminate the client-side connections and take out spurious retransmission from the server from the bill. In our experiment, however, we find that one can still launch attacks with port 80 in one of the U.S. ISPs, and all of them do not terminate the connections with port 8000. This implies that the attackers can use a non-standard HTTP port to launch TCP retransmission attacks. Amrutkar et. al. suggest a multidimensional diagnosis technique to find the root cause of poor end-to-end delays in a 3G network [42]. Their technique automatically pinpoints the root cause for the poor performance among applications, devices, and networks. They find that a particular device model can be a culprit of a high latency and that a gaming site was found to be the root cause for high latencies to a large number of subscribers.

Previous works have pointed out security problems with middleboxes (e.g., NATs or firewalls) in cellular networks. Wang et. al. find that four cellular ISPs allow IP spoofing such that a malicious client punches a hole in the NAT with an IP address of a victim, through which a colluding server blasts attack traffic [43]. This implies that an adversary can launch the "usage-inflation" attack even without redirecting the victim to a malicious site. They also find that some firewalls do not remove the flow state even after connection closure, which is confirmed in our experiments as well. Qian et. al. introduce a new attack where an off-path attacker (not man-in-the-middle) hijacks a TCP connection and injects malicious content [44]. Again, this can be exploited in the context of the usage-inflation attack without the knowledge of the victim.

B. Cellular Data Accounting

Peng et. al. have revealed that attackers can exploit a few policy loopholes in cellular accounting systems [7], [8]. The "toll-free data access attack" allows the attackers to bypass cellular accounting by using the DNS port whose data usage is not accounted by some ISPs. The authors also introduce a "stealth-spam attack" where a malicious attacker injects a large volume of spam data after the client closes the connection. While we observe the same outcomes in our work (e.g., free usage or usage inflation), our attacks stand on the fundamental difficulty in inferring the root cause of TCP retransmission in the middlebox. In comparison, the DNS port abuse seems to be easy to block as we notice that cellular ISPs in the U.S. no longer allow DNS port tunneling [32].

Tu et. al. find the cellular data accounting gap in as the user moves from one network to another [9]. They see as much as 69.6% of accounting disparity when they use UDP packets while roaming since the previously buffered packets in one network would drop when inter-system or intra-system handoffs occur (e.g., 2G↔3G, 3G↔LTE). However, we suspect the accounting disparity is not severe in real life given the heavy dominance of TCP in cellular networks [5], [6].

Raj et. al. propose splitting the cellular data bill by having the content providers pay for the traffic generated by the users visiting their websites [45]. They suggest accounting the cellular data in the client devices using ARM TrustZone [46]. We think data accounting in the client side could be extended to mitigate both TCP retransmission attacks by preventing "free-riding" attack from the client and by reporting the re-transmission amount to the cellular ISPs. However, deployment of such a scheme could be challenging since it requires modification of all client devices. Instead, we mainly focus on the middlebox-based solution in this work.

Earlier, we have measured TCP retransmission accounting policies with five cellular ISPs in the U.S., and South Korea, and revealed the vulnerabilities in the accounting systems [32]. In this work, we extended the measurements to 12 ISPs in

6 countries, experimented with the attacks in various real-world environments, showed the practicality of "free-riding" attacks, and discussed the solution space in depth. We have also designed and implemented Abacus, a highly-scalable accounting system, and showed that one can detect "free-riding" attacks on a low-powered desktop machine even when there are 100Ks of concurrent flows.

VIII. Conclusion

In this paper, we have shown that current cellular accounting systems are vulnerable to TCP retransmission attacks without proper defense mechanisms in place. We characterized 12 ISPs in 6 countries and demonstrated that an attacker can arbitrarily inflate the data usage without the knowledge of the target users or the attacker can tunnel data for free over TCP retransmissions, depending on the policies of the ISPs. The "usage-inflation" attack is launched from a remote server and does not require modification of the target devices. We have also shown that the "free-riding" attack is practical such that the attacker can use 15.6 to 22.1 Mbps of free bandwidth.

The fundamental challenge in defending these attacks stems from the fact that middlebox-based accounting systems cannot reliably infer the states of two TCP end nodes. We think that it is difficult to defend against the "usage-inflation" attack without false positives, and propose to detect the "free-riding" attack with a "selective" accounting policy. As a solution, we have implemented and evaluated Abacus that effectively manages 100Ks of concurrent flows with a small amount of memory and CPU usage while reliably detecting the "free-riding" attack.

IX. Acknowledgment

We would like to thank Nico Golde, Keon Jang, Haewoon Kwak and Qiuyu Xiao for providing great help in carrying out the retransmission accounting policy measurement tests. We also appreciate many insightful comments from anonymous reviewers. This research was supported, in part, by the National Research Foundation of Korea (NRF) grant #2012R1A1A1015222, the U.S. Department of Health and Human Services (SHARPS program) under Award Number 90TR0003-01, and the TerraSwarm Research Center, one of six centers supported by the STARnet phase of the Focus Center Research Program (FCRP) a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

References

[1] United Nations International Telecommunication Union Aggregate Data. http://www.itu.int/en/ITU-D/Statistics/Documents/statistics/2013/ITU_Key_2005-2013_ICT_data.xls, 2013.

[2] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581 (Proposed Standard), April 1999. Obsoleted by RFC 5681, updated by RFC 3390.

[3] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 5681 (Draft Standard), September 2009.

[4] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[5] S. Woo, E. Jeong, S. Park, J. Lee, S. Ihm, and K. Park. Comparison of Caching Strategies in Modern Cellular Backhaul Networks. In *Proceedings of the International Conference on Mobile Systems, Applications and Services (MobiSys)*, 2013.

[6] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, Z. M. Mao, S. Sen, and O. Spatscheck. An In-depth Study of LTE: Effect of Network Protocol and Application Behavior on Performance. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2013.

[7] C. Peng, G. Tu, C. Li, and S. Lu. Can We Pay for What We Get in 3G Data Access? In *Proceedings of the Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2012.

[8] C. Peng, C. Li, G. Tu, S. Lu, and L. Zhang. Mobile Data Charging: New Attacks and Countermeasures. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2012.

[9] G. Tu, C. Peng, C. Li, X. Ma, H. Wang, T. Wang, and S. Lu. Accounting for Roaming Users on Mobile Data Access: Issues and Root Causes. In *Proceedings of the International Conference on Mobile Systems, Applications and Services (MobiSys)*, 2013.

[10] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. In *IEEE/ACM Transactions on Networking (TON)*, volume 7(4), pp. 458-472, 1999.

[11] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. TCP Congestion Control with a Misbehaving Receiver. In *ACM SIGCOMM Computer Communication Review (CCR)*, volume 29(5), pp. 71-78, 1999.

[12] A. Kuzmanovic and E. W. Knightly. Receiver-centric congestion control with a misbehaving receiver: Vulnerabilities and end-point solutions. In *The International Journal of Computer and Telecommunications Networking*, volume 51(10), pp. 2717-2737, 2007.

[13] 3GPP. Universal Mobile Telecommunications System. http://www.3gpp.org/Technologies/Keywords-Acronyms/article/umts.

[14] 3GPP. LTE. http://www.3gpp.org/LTE/.

[15] 3GPP. ETSI TS 25.410. 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; UTRAN Iu Interface: general aspects and principles.

[16] 3GPP. ETSI TS 32.200. Telecommmunication management; Charging management; Charging principles.

[17] 3GPP. ETSI TS 32.215. Telecommunication management; Charging management; Charging data description for the PS domain.

[18] 3GPP. ETSI TS 32.251. 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Telecommunication management; Charging management; Packet Switched (PS) domain charging.

[19] 3GPP. ETSI TS 29.060. General Packet Radio Service; GPRS Tunneling Protocol across the Gn and Gp interface.

[20] 3GPP. ETSI TS 23.060. General Packet Radio Service (GPRS); Service description; Stage 2.

[21] 3GPP. ETSI TS 32.240. 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Telecommunication management; Charging management; Charging architecture and principles.

[22] G. Fairhurst and L. Wood. Advice to link designers on link Automatic Repeat reQuest (ARQ), 2002.

[23] H. Inamura, G. Montenegro, R. Ludwig, A. Gurtov, and F. Khafizov. TCP over Second (2.5G) and Third (3G) Generation Wireless Networks, 2003.

[24] J. Gettys. Bufferbloat: Dark Buffers in the Internet. In *IEEE Internet Computing*, volume 15(3):96, 2011.

[25] H. Jiang, Y. Wang, K. Lee, and I. Rhee. Tackling Bufferbloat in 3G/4G Networks. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement Conference (IMC)*, 2012.

[26] Amazon web services. Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/.

[27] GNU Operating System. Wget. http://www.gnu.org/software/wget/.

[28] Gadgetcat. tcpdump on Android, 2011. http://gadgetcat.wordpress.com/2011/09/11/tcpdump-on-android/.

[29] n1mda-dev. Pirni native iPhone ARP spoofer and network sniffer. http://code.google.com/p/n1mda-dev/wiki/PirniUsageGuide.

[30] KakaoTalk. http://www.kakao.com/talk/en/.

[31] S. Alcock and R. Nelson. Application flow control in youtube video streams. *ACM SIGCOMM Computer Communication Review*, 41(2):24–30, 2011.

[32] Y. Go, D. F. Kune, S. Woo, K. Park, and Y. Kim. Towards Accurate Accounting of Cellular Data for TCP Retransmission. In *Proceedings of the ACM International Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2013.

[33] O. Dondurmacioglu. Top-5 Bandwidth Hungry Mobile Apps on a Wireless LAN. http://www.arubanetworks.com/blogs/top-5-bandwidth-hungry-mobile-apps-on-a-wireless-lan/.

[34] M. Krasnyansky and M. Yevmenkin. Universal TUN/TAP driver. http://vtun.sourceforge.net/tun.

[35] Wikipedia. RC4. http://en.wikipedia.org/wiki/RC4.

[36] M. F. X. J. Oberhumer. LZO. http://www.oberhumer.com/opensource/lzo/.

[37] Ookla. Speedtest. http://www.speedtest.net.

[38] M. Mahemoff. Webwait.com. http://webwait.com.

[39] S. Han, K. Jang, K. Park, and S. Moon. PacketShader: a GPU-accelerated Software Router. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2010.

[40] A. Partow. General Purpose Hash Function Algorithms. http://www.partow.net/programming/hashfunctions/index.html.

[41] S. Han, S. Marshall, B. Chun, and S. Ratnasamy. MegaPipe: A New Programming Interface for Scalable Network I/O. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2012.

[42] C. Amrutkar, M. Hiltunen, T. Jim, K. Joshi, O. Spatscheck, P. Traynor, and S. Venkataraman. Why is My Smartphone Slow? On The Fly Diagnosis of Poor Performance on the Mobile Internet. In *Proceedings of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2013.

[43] Z. Wang, Z. Qian, Qiang Xu, Z. M. Mao, and M. Zhang. An Untold Story of Middleboxes in Cellular Networks. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2011.

[44] Z. Qian and Z. M. Mao. FlowTags: Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions. In *Proceedings of the IEEE Security and Privacy (Oakland)*, 2012.

[45] H. Raj, S. Saroiu, A. Wolman, and J. Padhye. Splitting the Bill for Mobile Data with SIMlets. In *Proceedings of the ACM International Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2013.

[46] ARM Security Technology. Building a Secure System using TrustZone Technology. ARM Technical White Paper, 2005–2009.