Linux Implementation and Evaluation of a Cooperation Mechanism for Hybrid Wireless Networks

Attila Weyland, Torsten Braun, Thomas Staub
Institute of Computer Science and
Applied Mathematics, University of Bern
Neubrückstrasse 10, 3012 Bern, Switzerland
{weyland|braun|staub}@iam.unibe.ch

Carolin Latze
Department of Informatics
University of Fribourg
Bd de Pérolles 90, 1700 Fribourg, Switzerland
carolin.latze@unifr.ch

Abstract

Communication over multiple hops, such as in hybrid wireless networks, can only work if the individual hops cooperate by forwarding packets from other hops. We present the Linux implementation of our previously proposed cooperation and accounting strategy for hybrid wireless networks called CASHnet. We describe our implementation as well as our testbed, where we performed different evaluations regarding introduced delay and packet processing time. We identify the limitations of our scheme in a real-life scenario and discuss possible improvements.

1. Introduction

Hybrid wireless networks are also called multihop cellular networks and were first proposed in [13]. Hybrid wireless networks consist of base stations and mobile stations. Multi-hop communication is used to extend the coverage of the base stations. Hybrid wireless networks combine the flexibility of mobile ad hoc networks and the reliability of infrastructure-based wireless networks. They are a promising concept for wireless Internet service providers, because they allow to dynamically extend the network coverage of hotspots, without deploying additional base stations. The hybrid wireless network architecture forms also part of the mesh network conept. However, numerous issues known from mobile ad hoc networks need to be addressed, such as security and cooperation.

Cooperation in hybrid wireless networks has received considerable attention over the past years. Numerous concepts were proposed, in order to ensure that network participants do not only transmit self-generated packets, but also forward packets from other nodes. We distinguish between cooperation enforcement and encouragement schemes. With cooperation enforcement nodes are threatened with (partial) exclusion from the network in case of uncooperative behavior [2, 6, 9, 14–16, 18, 25]. Usually, enforcement schemes rely on neighborhood monitoring and reputation systems. Their challenges include the correct detection of (un)cooperative behavior and the assurance of the efficiency of the (threat of) punishment. In cooperation encouragement schemes nodes are rewarded with virtual money in case of cooperativeness, as for example in [4,5,7,10,11,26]. They typically rely on tamper resistant hardware in order to protect charging and rewarding mechanisms and some central security/accounting instance. Their challenges are to find the balance between security and efficiency of the charging and rewarding mechanisms as well as to ensure the control over the virtual cash flow.

CASHnet [23], our cooperation and accounting strategy in hybrid wireless networks, is an encouragement scheme with a hybrid (partly centralized and decentralized) accounting architecture. Compared to fully decentralized accounting schemes, CASHnet ensures a constant cash flow and keeps the provider in control of it as we showed in [24]. Unlike completely centralized accounting schemes, CASHnet does not put additional signaling load on the links toward the base stations. In order to analyze the processing overhead under real conditions with real computers and networks, we implemented a prototype of our cooperation scheme under Linux. In Section 2 we describe the operation of CASHnet. Section 3 presents the implementation architecture, the operation as well as the challenges we encountered. Our testbed setup is explained in Section 4. The results from our evaluations follow in Section 5. Finally, we summarize our work.

2. CASHnet

CASHnet introduces charges and rewards to the packet transmission process. Packet originators as well as final recipients located in different multihop cellular networks get charged on the node for the transmission and the recep-

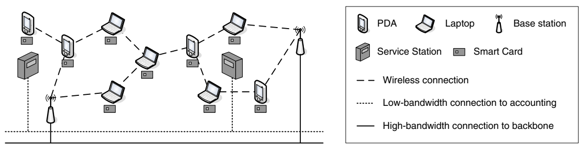

PDA
Laptop
Base station
Service Station
Smart Card
Wireless connection
Low-bandwidth connection to accounting
High-bandwidth connection to backbone

Figure 1: CASHnet example scenario.

tion of each packet respectively. The cost can be dynamic, i.e., related to the hop count to the base station or can be a globally fixed. Intermediate nodes forwarding packets are rewarded by their next hop on the route towards the packet's final destination. The frequency of the transmission of rewards can be adjusted, e.g., only every 10th forwarded packet gets acknowledged by the next hop. Due to the rewarding among nodes, source routing is not required. CASHnet does not account for ad hoc only traffic, i.e., packets with the final destination located in the same multihop cellular network as the originator get neither charged nor rewarded.

CASHnet is - to the best of our knowledge - the first cooperation encouragement scheme with a hybrid accounting architecture. It retains the flexibility of the multihop communication paradigm and ensures the network provider's complete control over the cash flow. CASHnet uses two virtual currencies, one for charging (traffic credits, TC) and one for rewarding (helper credits, HC). The purchase and exchange of virtual currencies can only be performed with the network provider or its representative. The decentralized accounting component consists of charging on each node and rewarding among nodes. The centralized accounting component is represented by the supervised refill of the virtual currency account at so called service stations operated by a network provider. A service station is similar to a low-cost terminal for loading prepaid cards and has a secure, low-bandwidth connection to the network provider for authentication and payment operations.

A node's role (packet originator, forwarder or final recipient) in the packet transceiving process is protected by a digital signature. Certificates with short lifetime issued by the network provider ensure the regular visit of nodes at the service stations, where possible misbehavior can be disciplined. All credentials and the virtual currency accounts are stored on a smart card issued by the network provider.

Fig. 1 depicts an example scenario for CASHnet. It shows a multihop cellular network with several mobile nodes equipped with smart cards, two interconnected service stations and two interconnected base stations also called gateways. The service stations and the base stations are connected to their respective backbone network.

A typical sequence of actions for a customer, who wants to participate in a CASHnet-enabled multihop cellular network, consists of 5 steps: preparation, authentication, transmission/reception & charging, forwarding & rewarding as well as refill. The first and the last step are performed at the service station, where the customer inserts her smart card. Fig. 2 illustrates step 2-4 in a message sequence chart in our example scenario. The numbered gray markers refer to example positions of the actions from the following list.

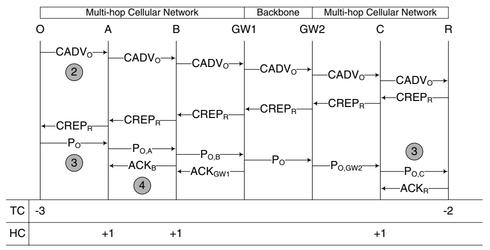


Figure 2: CASHnet operation with both originator and recipient in a MCN.

1. Preparation: The customer obtains the smart card from the network provider and loads the traffic credits accounts at the service station.
2. Authentication: Prior to the normal communication with a recipient, originator O sends a certificate advertisement $CADV_O$ to recipient R. All intermediate nodes (A, B and C) and the recipient will obtain the authentication information of the originator. The recipient in turn replies with a certificate reply $CREP_R$ addressed to O. All intermediate nodes will obtain the authentication information of the recipient.
3. Transmission / Reception & Charging: Prior to the transmission of a self-generated packet, the originator's traffic credit account is charged and the packet is digitally signed. Upon arrival of a packet at its destination, the recipient's traffic credits account is also charged.
4. Forwarding & Rewarding: At the reception of a packet, the node rewards the previous forwarding node in case it was not the originator or a gateway by sending a digitally signed acknowledgment ACK immediately or after receiving several forwarded packets. Receiving an ACK increases the node's helper credits account. The node also removes the digital signature of the previous node and adds its own before forwarding the packet. In addition, the node keeps the digital signature of each forwarded packet in order to validate the $ACKs$ sent by the next forwarding node.
5. Refill: After some time, the customer goes to a service station in order to refill his traffic credit account by exchanging available helper credits and/or buying traffic credits for real money.

For an in-depth description on the CASHnet operation and security mechanisms, we refer to [22].

3. Implementation

CASHnet affects the handling of each received, forwarded or generated packet. It provides a network layer service, i.e., it encourages forwarding of packets, by charging traffic generators and rewarding the forwarders. Thus, we require access to each packet after it enters and before it leaves a node. In addition, CASHnet requires cross-layer knowledge (i.e., security and accounting information). We decided to use Linux as development environment, because it provides great flexibility in accessing packets on the network stack.

3.1. Architecture

We implemented CASHnet as a user space daemon instead of a kernel module and accept a speed penalty caused by the additional communication between kernel and user space. However, we avoid the complexity and the rigidity of the monolithic Linux kernel. As a user space daemon, we have no permission to directly access the packets on the network stack. Therefore, we require the help of a program which establishes a bridge between kernel and user space. We use the netfilter/iptables [20] package to perform this task.

Netfilter and iptables are building blocks of a packet processing framework for the Linux kernel versions 2.4 and 2.6. Netfilter/iptables is widely used as firewall and for network address translation (NAT). Netfilter provides a set of hooks inside the network stack of the Linux kernel, allowing a kernel module to register callback functions. Fig. 3 shows the different netfilter hooks and their location in the Linux kernel network stack as well as three example packet flows. A registered function is called every time a packet traverses the respective hook in the network stack. Also, each hook allows modifying packets. Iptables provides a generic table structure for the definition of rule sets. A rule within iptables consists of a number of classifiers and an associated action, which is executed when a packet matches the classifiers. Thus, iptables allows controlling the packet flow and netfilter provides the required access to the network stack of the Linux kernel. Table 1 lists the tables, their chains and the hooks respectively.

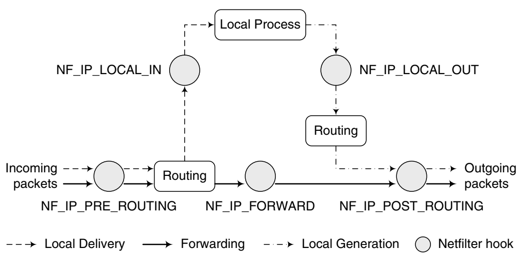


Figure 3: Netfilter hooks.

Table	Chain	Netfilter Hook
filter	INPUT	NF_IP_LOCAL_IN
	FORWARD	NF_IP_FORWARD
	OUTPUT	NF_IP_LOCAL_OUT
nat	PREROUTING	NF_IP_PRE_ROUTING
	OUTPUT	NF_IP_LOCAL_OUT
	POSTROUTING	NF_IP_POST_ROUTING
mangle	PREROUTING	NF_IP_PRE_ROUTING
	OUTPUT	NF_IP_LOCAL_OUT

Table 1: Tables, chains and hooks in iptables/netfilter.

We developed CASHnet in C++ under Linux using the GNU Compiler Collection, GCC as well as the libipq library from netfilter/iptables. Initially, we used smart cards from Axalto [3] called Cryptoflex 32K with an e-gate USB interface. OpenSC [1] provides the API for our program. However, due to the high delays introduced by the smart cards, we decided to use the RSA reference implementation called RSAREF [19] for the cryptographic functionality such as the creation and verification of digital signatures. As netfilter operates on the network layer, we deal with IP packets. In particular, we add, remove and verify digital signatures in the IP packet payload. For the test environment we require a mobile ad hoc routing protocol with gateway functionality. We chose AODV-UU [17], which implements the routing logic in a user space daemon and uses netfilter/iptables to access the packets.

Fig. 4 shows the interaction of CASHnet and netfilter/iptables in the context of the Linux system. CASHnet runs in user space and has no direct access to the network stack of the Linux kernel. In order to access all locally received, forwarded and generated packets, we use the respective chains of the filter table from iptables (INPUT, FORWARD and OUTPUT), which in turn use the respective netfilter hooks. All packets traversing these hooks are sent to the QUEUE target, a buffer in user space. The libipq library allows us to initialize this queue, to manipulate each packet as well as to decide about accepting or dropping. In addition, we use an UDP socket for the generation of the CASHnet control messages, i.e., the certificate advertisement *CADV* and reply *CREP* as well as the acknowledgment *ACK*. We receive these control messages by intercepting them via the INPUT chain, because netfilter had difficulties in delivering them to the UDP socket.

3.2. Operation

In the following, we describe the main procedures according to the different packet types handled by CASHnet. We distinguish between AODV, CASHnet control (*CADV*, *CREP* and *ACK*) as well as data packets. Although, the packets are passed to the same QUEUE, libipq allows us to find out via which iptables chain they entered. AODV packets are immediately accepted and not further processed.

When we receive CASHnet control messages three possibilities exist. For an acknowledgment *ACK* destined for

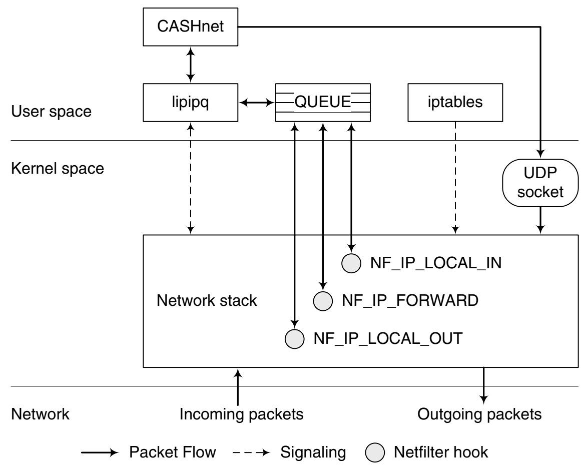


Figure 4: CASHnet and netfilter/iptables interaction in the Linux system.

the current node we reward the node, otherwise the message is forwarded. When a certificate advertisement *CADV* is intercepted, we add the certificate to the list of authenticated nodes. If the current node is the destination, we generate a certificate reply *CREP*. In case a *CREP* is filtered, we add the certificate to the list of authenticated nodes. If the current node is not the destination, we forward the certificate reply.

In case we intercept a data packet, also three possibilities exist. When the current node generated this packet, we charge the node's account as well as sign and transmit the packet. If the current node is the destination, we also charge the node as well as remove all additional data and deliver the payload to the local process. In case neither is true, the current node is forwarding the packet. Thus, we remove the signature from the previous hop and add one from the current node.

3.3. Challenges

CASHnet operates on the network layer and adds additional information (e.g. signature, nonce) to each IP packet. This information is part of the IP payload and would be destroyed by fragmentation. For TCP, netfilter provides the possibility to set the MTU accordingly by adjusting the maximum segment size parameter. In contrast, UDP applications must be aware of this parameter and generate packets with smaller payload.

When a node communicates with a correspondent outside its current multihop cellular network, AODV-UU establishes a tunnel between that node and the selected gateway for the duration of the communication. According to the authors this is more robust than relying on the default route in case of topology changes. In order to tunnel a packet AODV-UU intercepts it via netfilter. It does so after CASHnet has digitally signed the packet, making it impossible for us to verify the signature correctly at intermediate nodes. In order to avoid tunneling of AODV-UU, we had to restrict the test range from laptop A to the laptop operating as gateway. Fortunately, this does not affect our evaluations, as the gateway performs the verification and removal of the signature as if a backbone host would be the recipient. The only difference is that instead of forwarding the packet, it is delivered to the local process. For further details on the CASHnet implementation process under Linux, we refer to [12].

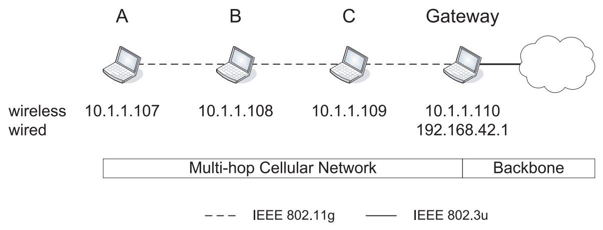


Figure 5: Testbed.

4. Testbed Implementation

The CASHnet framework consists of different components and functionalities. A CASHnet node runs a daemon, which is responsible for charging the generated traffic and rewarding the forwarded traffic. The daemon also exchanges certificates for the authentication and is responsible for the creation and verification of digital signatures.

In order to evaluate our CASHnet implementation we set up a small testbed shown in Fig. 5. In this testbed four laptops (A, B, C and Gateway) are interconnected in a chain topology. Node A and C have an Intel Celeron processor 2.40 GHz with 128 KB cache, node B has an Intel Pentium M processor 1.86 GHz with 2 MB cache and the gateway has an Intel Pentium M processor 1.4 GHz with 1 MB cache. The laptops all have an internal wireless network interface card compliant to IEEE 802.11g, which allows a maximum gross data rate of 54 Mbit/s. The gateway has an Ethernet link with 100 Mbit/s (IEEE 802.3u) to the backbone. All laptops run Slackware Linux 10.1 [21] with the Linux Kernel 2.6.8, netfilter/iptables 1.2.11, NdisWrapper 1.2 [8] for the driver of the wireless cards and AODV-UU 0.9 [17].

To emulate a chain topology in our laboratory room, we set up a MAC address filter such that each node only receives packets from its direct one-hop neighbor(s). On the gateway laptop we enable the gateway mode of AODV-UU and specify a locality netmask prefix to let AODV-UU distinguish the mobile ad hoc network from the normal network. For simplicity we use private addresses. On each laptop, we generate a public-/private-key pair as well as the corresponding digital certificate using RSA and MD5 for the hash generation. We use a key length of 1024 bits. We

	Scenario				
	1	2	3	4	5
Parameter	Plain	Pass	Partial	Full	PCAT
Source	A, B, C				A
Destination	Gateway				
Data signed	-	no	yes	yes	yes
ACK signed	-	no	no	yes	yes
PCAT	-	1	1	1	1,5,10,15,20

Table 2: Parameter settings for the testbed scenarios.

regard this as a reasonable trade-off between security and performance considering the short certificate lifetime on a node, which we set to 5 minutes in the test scenario.

We identified four variable key parameters for the evaluation of our CASHnet Linux implementation. These parameters have a strong influence on the CASHnet implementation and provide us with high flexibility when conducting performance measurements. The following list explains the parameters in detail.

- *Source:* We vary the source of our delay measurements between the four laptops. The distance between source and destination expressed in the number of intermediate hops strongly influences the end-to-end delay measurements, because each intermediate node processes the packet, i.e., it verifies and creates digital signatures. We vary the source of our measurements between laptop A, B and C.
- *Data/acknowledgments signed:* As mentioned before, we distinguish between different packet types. In order to measure the impact of the cryptographic functions, we allow specifying whether data packets and/or acknowledgments are digitally signed in the respective test scenario. This directly affects the processing time on the node, and thus the end-to-end delay between source and destination.
- *Packet counter ACK threshold:* The packet counter ACK threshold defines how many forwarding packets a node has to receive from a single forwarder before it rewards this forwarding node and sends an acknowledgment message. A node rewards a forwarding node after receiving 1, 5, 10, 15, or 20 packets.

From the variation of these parameters we created five test scenarios listed in Table 2. In the first scenario (plain) no CASHnet functionality is activated. Here, we obtain an indicator for the optimal performance of our testbed. In the second scenario (pass), we activate CASHnet, but do not use the security functionality. This scenario gives us information about the delay caused by processing the packet in user space. The third scenario (partial) allows us to measure the delay caused by digitally signing data packets. With the fourth scenario (full) we study the delay caused by the complete security operations of CASHnet, i.e., data and acknowledgments are digitally signed. In these four scenarios, we let each node ping the gateway consecutively, resulting in three sub scenarios for each main scenario. In the fifth scenario, we analyze the delay caused by different packet counter ACK thresholds (PCAT).

5. Evaluation

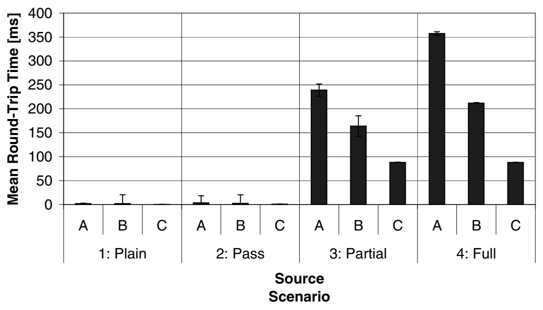
With the Linux implementation of CASHnet, we evaluate the end-to-end delay and jitter as well as the packet processing time on each node. For simplicity, we deactivate the accounting functionality of CASHnet. In our tests, we use ping to transmit 2000 packets at a rate of 1 packet per second.

We note that in a real test environment the number of possible impacts is huge and phenomena are difficult to isolate. During our measurements we found that simple channel scanning on the wireless medium by other computers greatly affected the performance. Also, we measured sporadic outliers independent of the scenario, which we attribute to retransmission on the MAC layer or possible variations in the processing time of intermediate nodes. The first is typically caused by transmission errors on the wireless medium, the latter by the packet processing applications, e.g., CASHnet and AODV-UU. Nevertheless, the results give some indication how CASHnet performs in a real environment and show possible improvements.

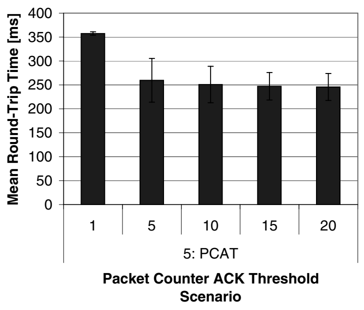
5.1. Round-Trip Time

The round-trip time is an indicator for the end-to-end delay and jitter between two communication partners. We measure it by analyzing the output from the ping program in each testbed scenario. Fig. 6 presents the mean round-trip time as well as the standard deviation. The expected behavior would be a round-trip time decrease when approaching the gateway in the sub scenarios and an increase for main scenarios, when including more and more security functionality. As expected first Plain scenario shows the best performance with average round-trip times below 2 ms, for both node A and B, and 1 ms for node C pinging the gateway. In the Pass scenario, all packets from the respective chains are processed by CASHnet in user space with security functionality disabled. Therefore, the average round-trip times slightly increase by approx. 1-2 ms compared to the Plain scenario. When we start to apply a part of the security operations in the Partial scenario, i.e., digitally sign data packets, the round-trip times increase considerably. The round-trip times further increase when we sign data and acknowledgment packets in the Full scenario.

The delay on node C does not increase between the Partial and Full scenario, because in CASHnet the packet originator does not receive a reward. Since there is no intermediate forwarding hop between node C and the gateway (C is the originator of the echo requests and the gateway



(a) Scenario 1 - 4: Variation of enabled CASHnet functionality between none (plain), pass, partial and full when node A, B and C ping the gateway.



(b) Scenario 5: Variation of different packet counter ACK thresholds when node A pings the gateway.

Figure 6: Mean round-trip times and standard deviation for all testbed scenarios.

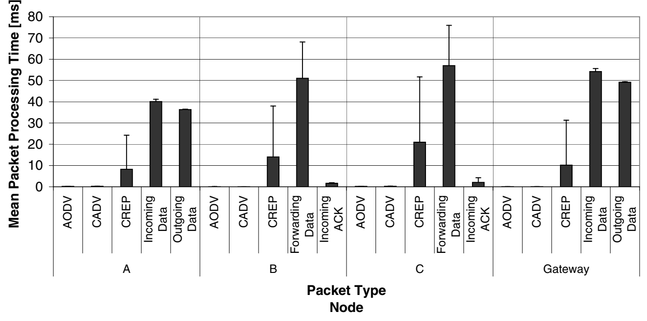
is the originator of the echo responses) no packets get acknowledged. When node B pings the gateway in the Full scenario, the gateway and node B send an acknowledgment to node C for every echo request and echo response respectively. The insertion of 1 intermediate hop and the resulting acknowledgments increase the delay by 41 ms compared to the Partial scenario. In case node A pings the gateway in the Full scenario, node C and the gateway acknowledge every echo request to node B and C respectively. In addition, node A and B acknowledge every echo response to node B and C respectively. Again, 2 intermediate hops and acknowledgments increase the delay by 119 ms compared to the Partial scenario. The results of the PCAT scenario show a significant decrease in delay by 98 ms when we acknowledge every fifth instead of every single forwarded packet. The continuous increase of the packet counter ACK threshold decreases the delay in small steps, because another important cause for the delays remains unchanged: the number of intermediate hops, which have to digitally sign and verify each packet. In fact, the delay approaches the 239 ms measured in the Partial scenario, where we did not digitally sign any acknowledgment.

5.2. Packet Processing Time

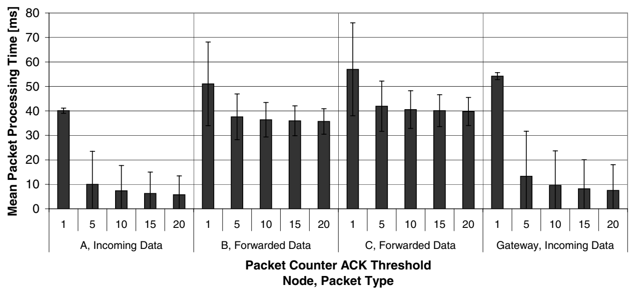
The packet processing time describes the amount of time a packet spends inside the CASHnet implementation. We measure it by logging each packet with a timestamp, when it enters and exits the CASHnet implementation. Because CASHnet uses public key cryptography the processing can take quite long time. We do not measure the additional overhead resulting from the communication between user and kernel space. We distinguish data packets according to the chain via which we intercepted them, i.e., incoming, forwarded or outgoing data. In addition, we gather the processing time for acknowledgments received. Moreover, we monitor the processing time of certificate advertisements *CADV*, certificate replies *CREP* and *AODV* packets. Fig. 7 shows the average packet processing time for all nodes in the PCAT scenario, where node A pings the gateway under different packet counter ACK thresholds (PCAT = 1, 5, 10, 15, or 20). In this scenario node A generates the echo requests and the gateway generates the echo responses. Both nodes do not receive any acknowledgments because they are the endpoints of the bidirectional communication. Nodes B and C are intermediate forwarding nodes and therefore receive acknowledgments. Fig. 7(a) shows the processing time for all packets on the respective nodes in the case, where we acknowledge each packet (PCAT = 1). For other PCAT, only some packet types showed considerable changes as values presented in Fig. 7(b). Changes occur for the processing of the incoming but not for the outgoing data on node A and the gateway. We also measure changes for the forwarding data at intermediate nodes. In these measurements, the processing time for data packets includes the time to generate and transmit an acknowledgment.

From Fig. 7(a) we see the time it takes to process the different packet types on each node, when we acknowledge every packet in the PCAT scenario. On all nodes, we measure very low average processing times for AODV and certificate advertisement packets. Both vary between 50 and 200 μs. We can explain the variation with the impact of the operating system scheduler. As described in Section 3.2, AODV packets are immediately accepted, i.e., handed back to the network stack upon interception. Certificate advertisements are also immediately accepted unless the current node is the destination. In this case, the node creates a certificate reply.

In the PCAT scenario, node A is the originator of the echo requests. CASHnet requires 36 ms in average to digitally sign these packets on node A (outgoing data). Processing of this echo request at the gateway requires 54 ms (incoming data), which includes the time to generate an acknowledgment for the last forwarding node C. The gateway sends back an echo response and CASHnet takes 50 ms in average to digitally sign it (outgoing data). Node A requires 40 ms to process this response (incoming data). Again, this duration includes the generation of an acknowledgment for the last forwarding node B. We note that processing of both



(a) Node A, B, C and the gateway with PCAT = 1.



(b) Data packet processing at node A, B, C and the gateway for different PCAT.

Figure 7: Mean packet processing times and standard deviation of all nodes for the PCAT (packet counter ACK threshold) scenario.

incoming and outgoing data takes 14 ms more at the gateway than at node A. We explain this difference with different processor speeds on node A (2.4 GHz) and the gateway (1.4 GHz). The intermediate nodes B and C only forward data. Thus, they receive acknowledgments from their respective next hops. CASHnet on node B requires 52 ms to verify and digitally sign the data (echo requests and responses) as well as to acknowledge the echo responses forwarded by node C. The same processes take 57 ms on node C. Again, we attribute the difference to the distinct processors of node B and C. The first is a Pentium M with 2 MB cache, the latter a Celeron with 128 KB cache. The total processing delay for data packets in CASHnet on all nodes sums up to 289 ms. From the round-trip time measurements we obtained a delay of 358 ms. We explain the missing 69 ms with influences from outside CASHnet in the operating system or from the possible interaction of AODV and CASHnet while using netfilter. The verification of received acknowledgments (incoming ACK) lasts 2 ms on both nodes.

In addition, the time to process a certificate reply changes between the different nodes. Every 5 minutes or 300 pings, the certificates must be refreshed. Thus, the nodes send a certificate advertisement and answer with a certificate reply. The reception of a certificate reply causes processing (digital signature) of eventually queued packets. Depending on the number of packets in the queue this can cause a significant delay, which is included in the processing time of the certificate reply. From Fig. 7(a) we observe that the average processing delay at the sender and receiver nodes is lower than at the intermediate nodes. In contrast to node A and the gateway, intermediate nodes B and C receive acknowledgments for their forwarding and have to process them. Thus, in case a certificate becomes invalid more packets wait in the queue and must be processed when the certificate reply arrives, which in turn results in higher delays. As for the forwarding data, we attribute the difference between node B and C to the more powerful processor in node C.

Fig. 7(b) shows the average data packet processing time at all nodes in the PCAT scenario. Node A and the gateway generate and receive data. However, only incoming data shows a change in the processing time for different packet counter ACK thresholds. We see that an increase in the packet counter ACK threshold reduces processing time for the incoming data, e.g., from 40 to 10 ms at node A and from 54 to 13 ms at the gateway, when we acknowledge every fifth instead of every single packet. When CASHnet receives a packet destined to the current node, it rewards the forwarding node before it passes the packet to the local process. Thus, the reduced number of acknowledgments, which need to be transmitted for the same amount of data received, reduces the processing time for the incoming data packets. As in the previous round-trip time measurements we noticed a slow-down in the decrease for higher packet counter ACK thresholds (10, 15, 20), because the number of generated acknowledgments decreases slowly.

Because nodes B and C are intermediate nodes, they receive data packets to be forwarded. The processing time for forwarded data packets decreases from 51 to 38 ms at node B and from 57 to 42 ms at node C, when we increase the packet counter ACK threshold from 1 to 5. Compared to node A and the gateway the reduction is not so high, because the intermediate nodes still have to verify and digitally sign data packets in both directions. Node A and the gateway have no packets to forward and thus only need to digitally sign outgoing packets. In addition, the intermediate nodes have to process incoming acknowledgments from their one-hop neighbors, while node A and the gateway never receive acknowledgments.

The decentralized per-hop rewarding mechanism in CASHnet puts a higher work load on the intermediate nodes compared to the originator and recipient of a communication. For each data packet, an intermediate node must verify 2 digital signatures (originator and previous node) and compute a new one, while an originator only has to verify and compute 1 digital signature and a recipient only needs to verify 1 digital signature. Any node acknowledges a data packet, if the previous node was not the packet originator. An acknowledgment can proof the forwarding of more than one packet (PCAT > 1) and reduce the generated overhead. However, the expensive digital signatures for each data packet remain. While the digital signature of the packet originator is mandatory to ensure only authorized transmissions, the granularity of the digital signatures of the intermediate nodes could be increased. This would require the coordinated storage of many packet hashes at all nodes. Here, we see a trade-off between storage and computational power.

6. Summary

We implemented CASHnet under Linux and conducted several evaluations in a small testbed using laptops with wireless network interfaces in order to measure the end-to-end delay imposed by CASHnet. To support the identification of the cause for the delay, we analyzed the processing time for each packet type distinguished by the CASHnet implementation. We found the average round-trip time for a communication over 3 hops to be 358 ms. We could reduce this delay to 246 ms by acknowledging only every 20th packet. However, with a one-way transmission time of 123 ms over 3 hops, we are above desired delays. The round-trip time can be certainly improved by adjusting the security parameters, such as the algorithm and the key length.

The implementation helped us to adapt our CASHnet algorithm to a real environment and showed us possible limitations. The processing power of the nodes is a limiting factor, as the security functionality is computationally expensive. Also, a high variation in the processing power of the nodes causes certain nodes to take longer to process packets from queues, which in turn increases the jitter. We also found our test environment and in particular the processing power of the laptops as well as the interferences on the wireless medium to influence the results.

References

[1] Opensc smart card library #15 compatible cards. Website, 2006. [visited 17Feb.2006].

[2] T. Anker, D. Dolev, and B. Hod. Cooperative and reliable packet-forwarding on top of AODV. In *Proceedings of 4th WiOpt*, Boston, MA, USA, Apr. 2006. to appear.

[3] Axalto. Website, 2006. [visited 17Feb.2006].

[4] J. P. Barraca, S. Sargento, and R. L. Aguiar. The polynomial-assisted ad-hoc charging protocol. In *Proceedings of 10th ISCC*, pages 945–952, La Manga, Spain, June 2005.

[5] N. Ben Salem, L. Buttyán, J.-P. Hubaux, and M. Jakobsson. Node cooperation in hybrid ad hoc networks. *IEEE Transactions on Mobile Computing*, 5(4):377–387, Apr. 2006.

[6] S. Buchegger and J.-Y. Le Boudec. Performance analysis of the confidant protocol (cooperation of nodes - fairness in dynamic ad-hoc networks). In *Proceedings of 3rd Mobihoc*, pages 226–236, Lausanne, Switzerland, June 2002.

[7] L. Buttyán and J.-P. Hubaux. Stimulating cooperation in self-organizing mobile ad hoc networks. *ACM Mobile Networks & Applications*, 8(5):579–592, Oct. 2003.

[8] P. Fuchs and G. Pemmasani. Ndiswrapper, 2006. [visited 4Feb.2006].

[9] Q. He, D. Wu, and P. Khosla. Sori: A secure and objective reputation-based incentive scheme for ad-hoc networks. In *Proceedings of IEEE WCNC*, pages 13–15, Atlanta, GA, USA, Mar. 2004.

[10] M. Jakobsson, J.-P. Hubaux, and L. Buttyán. A micro-payment scheme encouraging collaboration in multi-hop cellular networks. In *Proceedings of 7th FC*, pages 15–33, Gosier, Guadeloupe, Jan. 2003.

[11] B. Lamparter, K. Paul, and D. Westhoff. Charging support for ad hoc stub networks. *Elsevier Journal of Computer Communications*, 26(13):1504–1514, Aug. 2003.

[12] C. Latze. Linux implementation of a cooperation and accounting strategy for multihop cellular networks. Master's thesis, University of Bern, Feb. 2006.

[13] Y.-D. Lin and Y.-C. Hsu. Multihop cellular: A new architecture for wireless communications. In *Proceedings of 19th Infocom*, pages 1273–1282, Tel Aviv, Israel, Mar. 2000.

[14] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan. Sustaining cooperation in multi-hop wireless networks. In *Proceedings of 2nd NSDI*, Boston, MA, USA, May 2005.

[15] S. Marti, T. J. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of 6th Mobicom*, pages 255–265, Boston, MA, USA, Aug. 2000.

[16] P. Michiardi and R. Molva. Core: A collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks. In *Proceedings of 6th CMS*, pages 107–121, Portoroz, Slovenia, Sept. 2002.

[17] E. Nordström. Aodv-uu, Dec. 2004. [visited 17Feb.2006].

[18] K. Paul and D. Westhoff. Context aware detection of selfish nodes in dsr based ad-hoc networks. In *Proceedings of IEEE GLOBECOM*, pages 178–182, Taipei, Taiwan, Nov. 2002.

[19] Rsa data security rsaref, 1993.

[20] P. Russell et al. netfilter/iptables, 2006. [visited 2Feb.2006].

[21] The Slackware Linux Project. Slackware linux, 2006. [visited 2Feb.2006].

[22] A. Weyland. *Cooperation and Accounting in Multi-Hop Cellular Networks*. PhD thesis, University of Bern, Nov. 2005.

[23] A. Weyland and T. Braun. Cooperation and accounting strategy for multi-hop cellular networks. In *Proceedings of 13th LANMAN*, pages 193–198, Mill Valley, CA, USA, Apr. 2004.

[24] A. Weyland, T. Staub, and T. Braun. Comparison of motivation-based cooperation mechanisms for hybrid wireless networks. *Elsevier Journal of Computer Communications*, 2006. to appear.

[25] H. Yang, X. Meng, and S. Lu. Self-organized network-layer security in mobile ad hoc networks. In *Proceedings of WiSe*, pages 11–20, Atlanta, GA, USA, Dec. 2002.

[26] S. Zhong, J. Chen, and Y. R. Yang. Sprite: A simple, cheat-proof, credit-based system for mobile ad-hoc networks. In *Proceedings of 22nd Infocom*, volume 3, pages 1987–1997, San Francisco, CA, USA, Mar.–Apr. 2003.