NOKIA

Research Center
NRC-TR-2008-002

TCP Wake-Up: Reducing Keep-Alive Traffic in Mobile IPv4 and IPsec NAT Traversal

Pasi Eronen
Nokia Research Center
pasi.eronen@nokia.com
January 31, 2008

Abstract:

Applications such as instant messaging and push email require long-lived connections between clients and servers. In the absence of other traffic, stateful firewalls and Network Address Translators (NATs) require "keep-alive" messages to maintain state for such persistent connections. We present new measurements analyzing the energy consumption of these keep-alive messages on a mobile phone in 2G (GSM), 3G (WCDMA), High-Speed Downlink Packet Access (HSDPA), and IEEE 802.11 Wireless LAN networks. The measurements confirm earlier results showing that frequent keep-alive messages consume significant amounts of energy in 2G and 3G networks, but suggest they are not a significant problem in Wireless LANs.

To reduce energy consumption, we introduce TCP Wake-Up, an extension to Mobile IPv4 and IPsec NAT traversal mechanisms. This extension significantly reduces the need for keep-alive messages, while still avoiding complexity of IP-over-TCP tunneling. Our measurements show that TCP Wake-Up can extend battery lifetime by a factor of 2 to 7 in 2G/3G networks. The results also suggest guidelines for developers of future protocols: in particular, we claim that "always-on" applications that aim to be used in current 2G/3G networks cannot be solely based on UDP.

Index Terms:
energy consumption
firewalls
Mobile IPv4
IPsec

TCP Wake-Up: Reducing Keep-Alive Traffic in Mobile IPv4 and IPsec NAT Traversal

Pasi Eronen
Nokia Research Center
pasi.eronen@nokia.com

***Abstract*—Applications such as instant messaging and push email require long-lived connections between clients and servers. In the absence of other traffic, stateful firewalls and Network Address Translators (NATs) require "keep-alive" messages to maintain state for such persistent connections. We present new measurements analyzing the energy consumption of these keep-alive messages on a mobile phone in 2G (GSM), 3G (WCDMA), High-Speed Downlink Packet Access (HSDPA), and IEEE 802.11 Wireless LAN networks. The measurements confirm earlier results showing that frequent keep-alive messages consume significant amounts of energy in 2G and 3G networks, but suggest they are not a significant problem in Wireless LANs.**

To reduce energy consumption, we introduce TCP Wake-Up, an extension to Mobile IPv4 and IPsec NAT traversal mechanisms. This extension significantly reduces the need for keep-alive messages, while still avoiding complexity of IP-over-TCP tunneling. Our measurements show that TCP Wake-Up can extend battery lifetime by a factor of 2 to 7 in 2G/3G networks. The results also suggest guidelines for developers of future protocols: in particular, we claim that "always-on" applications that aim to be used in current 2G/3G networks cannot be solely based on UDP.

I. Introduction

Firewalls and Network Address Translators (NATs) are ubiquitous in today's Internet, and it is increasingly rare to have a personal, non-server computer with global IP layer reachability. Instead, the host is protected by a stateful "middlebox" that keeps track of active connections, and drops packets coming from the "outside" unless they are part of an existing connection.

The firewall/NAT state is automatically created when the host "inside" the firewall/NAT (later called "client") initiates a connection, and it will be removed once the connection has been unused for some time. Because the connection state is created only by packets sent by the client, servers outside the firewall/NAT are not able to reach the client if the state has expired. To prevent this, many protocols regularly send dummy "keep-alive" packets that reset the timers in the NAT or firewall and preserve reachability.

The connection state timeout values vary from product to product, but typical values are 30…180 seconds for UDP and 30…60 minutes for TCP [8]. This implies that applications using UDP need to send keep-alive messages much more frequently than those based on TCP.

Most application layer protocols use TCP, but UDP is commonly used to allow network layer mobility and security mechanisms to co-exist with NATs and firewalls. In particular, the NAT/firewall traversal mechanisms for Mobile IPv4 [15] and IPsec [11] tunnel IP packets over UDP. Using UDP instead of TCP avoids the performance problems associated with multiple layers of TCP retransmissions [9][30] and delays due to head-of-line blocking.

Sending keep-alive messages can consume significant amounts of energy in small battery-powered devices. For example, Haverinen et al. [8] have shown that sending UDP keep-alives once every 40 seconds increases idle energy consumption in 3G WCDMA by a factor of 3 to 16 depending on the radio network configuration.

In Section II, we revisit the results of Haverinen et al. to examine the impact of phone model, new radio technologies (HSDPA and Wireless LAN), and the type of keep-alive message used. The results confirm that UDP keep-alives will lead to unacceptably short battery lifetimes in 2G, 3G, and HSDPA, but their impact is much smaller in Wireless LAN.

In Section III of this paper, we introduce TCP Wake-Up, an extension to Mobile IPv4 and IPsec NAT traversal mechanisms. The extension keeps the benefits of tunneling IP packets over UDP, but does not require sending UDP keep-alives. The basic idea is to establish a separate TCP connection between the client and home agent (or VPN gateway in the case of IPsec). When there is no ordinary data traffic, UDP keep-alives are not sent; thus, the connection state in the NAT or firewall will expire, and the client cannot be reached with UDP-tunneled packets. Instead, when the home agent needs to reach the client, it uses the TCP connection to "wake up" the client; the client will then re-establish the UDP-based tunnel.

All ordinary data packets are still sent using UDP encapsulation, avoiding the performance problems and complexity associated with TCP encapsulation. However, when the client is idle, the NAT mappings for UDP can be allowed to expire without losing reachability. Keep-alive messages are still needed for the TCP connection, but since the typical TCP mapping timeout is much larger than for UDP (at least one order of magnitude), the number of keep-alive messages, and thus energy consumption, is reduced. For example, in the 3G WCDMA network used for measurements in Section II, using TCP Wake-Up with typical NAT timeout parameters extends battery lifetime by a factor of 7. A more detailed analysis of the benefits and costs is presented in Section IV.

Compared to alternative solutions discussed in Section V, our work requires modifications only in the client and home agent/VPN gateway, but not in the network elements, such as radio access networks and NATs, between them. Thus, the solution can be deployed by, for example, enterprises that do not have control over the radio network.

II. Energy Consumption of Keep-Alive Messages

Haverinen et al. [8] have measured the energy consumption of keep-alive messages in 3G WCDMA networks. The results show that keep-alive messages are expensive in 3G chiefly because the radio channel stays allocated for a long time (at least several seconds) after the packet has been sent. This is because releasing the channel, and thus transitioning back to "low-power" state, is triggered by inactivity timers in the Radio Network Controller (RNC), not by the phone. Thus, configuration of RNC parameters has significant impact on the total energy consumption.

In this section, we present additional measurements with the following goals:

First, we want to find out if the results vary significantly from one phone model to another. The phone used by Haverinen et al., Nokia 6630, was announced in June 2004, and was Nokia's first 3G phone based on the Symbian operating system. It is plausible that hardware and software improvements have changed the situation since then.

Second, we want to examine additional radio technologies available in newer phones, including High-Speed Downlink Packet Access (HSDPA) and IEEE 802.11 Wireless LAN.

Third, we want to determine the effect of different types of keep-alive message exchanges. For example, in IPsec, keep-alive messages are sent only by the client, while in Mobile IPv4, the home agent sends back an acknowledgment.

A. Measurement Setup

The measurements were performed in Elisa 2G/3G/HSDPA network in Helsinki, Finland. Using a live network introduces aspects beyond our control (such as RNC configuration), but it may also give a more realistic picture than using a test network, as was done by Haverinen et al.

The phone used was Nokia E65 [17], except for HSDPA tests which were done with Nokia N95 [18]. All unneeded features of the phone, such as Bluetooth and display backlight, were turned off during the measurements.

To measure the energy consumption, we connected the phone to a power supply using a "dummy battery" and a high-precision 0.1 ohm shunt resistor. The dummy battery is an adapter commonly used in phone certification testing: it emulates the signaling provided by real batteries, and contains capacitors to smooth sudden voltage spikes.

Measurements were recorded using a National Instruments DAQPad 6015 data acquisition unit, connected to PC over USB. The voltage drop across the shunt resistor was sampled at 100 kHz with custom LabVIEW-based software, averaged over 100 ms intervals, and post-processed in Excel.

The software consisted of a Java MIDP application that sent keep-alive packets at configurable intervals, and corresponding server software for sending reply packets when needed.

B. Phone Model and Keep-alive Type

To determine the impact of phone model and keep-alive type, we measured the average power with two different types of keep-alive exchanges (a single unacknowledged UDP packet, similar to IPsec, and a two-message exchange with an acknowledgment packet, similar to Mobile IPv4), with both 2G and 3G networks. The measurements were repeated with several different keep-alive intervals (20, 40, 60, and 150 seconds), and without keep-alive messages.

A trace from typical measurement is shown in Fig. 1, with a detail magnified in Fig. 2. Fig. 2 shows clearly the transition from idle state to CELL_DCH, staying in CELL_DCH for 6…7 seconds ("T1" timer), transitioning to CELL_FACH for 2 seconds ("T2" timer), and eventually back to idle. (See [8] for a more detailed description of the state transitions and associated timers.)

Fig. 3 shows the results for 3G with acknowledgment packet with different keep-alive intervals. As expected, the result fit nicely on a line, and a least-squares fitting results in energy consumption of 9200 mJ/keep-alive exchange, plus 29 mW background consumption.

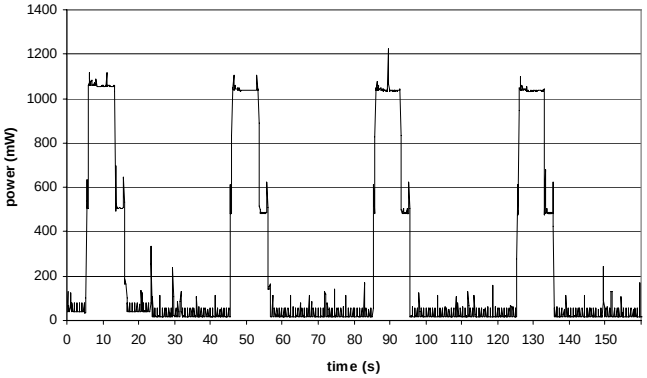

Fig. 1: Power consumption trace of keep-alive messages (with acknowledgment) in 3G with 40-second interval.

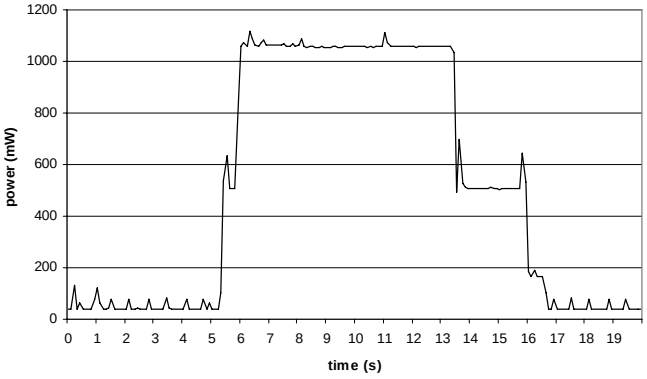

Fig. 2: Detail from Fig. 1, showing the time spent in different RRC states.

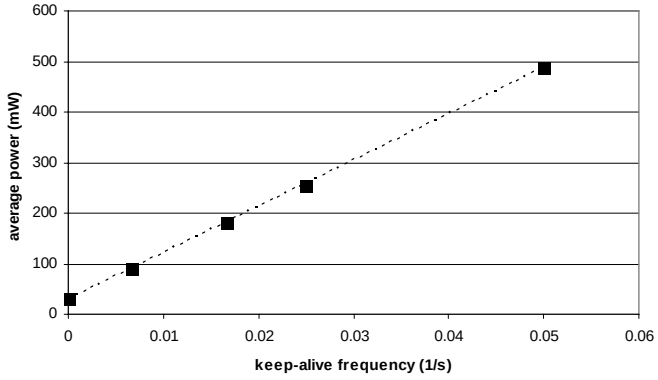

Fig. 3: Average 3G power consumption of keep-alive messages (with acknowledgment) with different intervals. The least-squares fitted line corresponds to 9200 mJ/keep-alive plus 29mW background consumption.

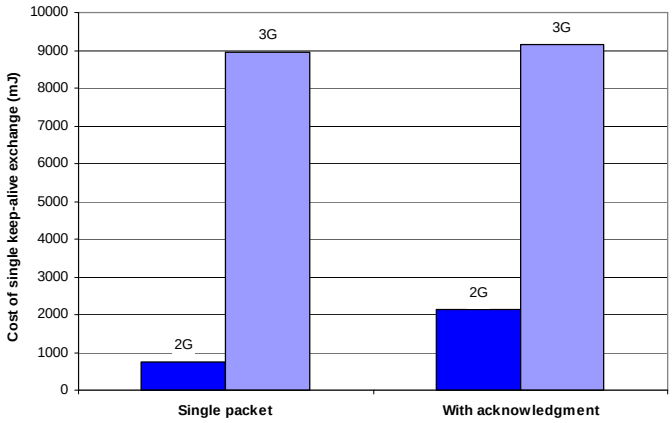

Fig. 4: Energy consumed by a single keep-alive exchange, for both 2G and 3G, with and without acknowledgment packet.

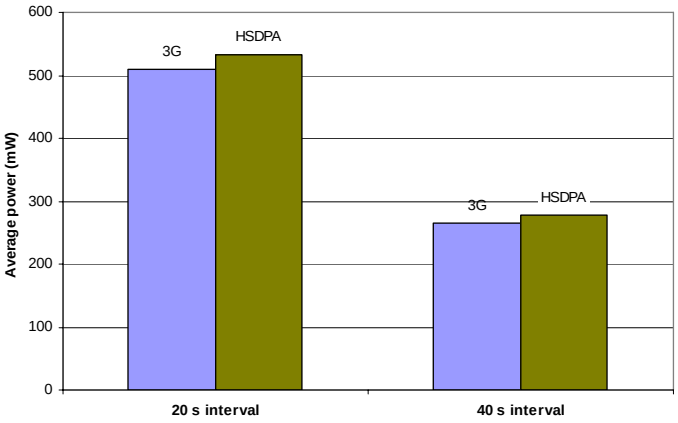

Fig. 5: Average power consumption of a keep-alive exchange (with acknowledgment) on Nokia N95, comparing 3G (without HSDPA) with HSDPA.

Fig. 4 summarizes the results of least-squares fitting for the other combinations. The background consumption in 2G was 16…19 mW, significantly less than in 3G. With this particular RNC configuration, the cost of a single keep-alive exchange was also much smaller in 2G.

These results are reasonably in line with earlier measurements. For 3G network (with T2=2 s and CELL_PCH disabled), Haverinen et al. estimate that a single keep-alive exchange consumes 0.61 mAh (8100 mJ at nominal battery voltage of 3.7 V), with background current of 6.1 mA (23 mW). We can thus conclude that the impact of phone model is small in 2G/3G networks compared to the network parameters and keep-alive interval.

An interesting result is that in 2G, a keep-alive exchange with acknowledgment packet consumed almost three times as much energy as a single packet. Apparently, the presence of downlink packet led to different state transitions in this particular network configuration. For 3G, there was no significant difference.

We also tested a simple application-layer keep-alive exchange over TCP; although this results in three packets (the client has to send TCP ACK last), the energy consumption was basically the same as with two-packet exchange.

C. HSDPA

The second set of measurement investigated the impact of HSDPA. We measured the power consumption of a two-message keep-alive exchange with plain 3G and HSDPA, using two different keep-alive intervals (20 and 40 seconds). Fig. 5 shows the average power in these four cases.

In this particular network setup, there does not seem to be significant difference between plain 3G and HSDPA. The phone display indicators also suggested that HSDPA channels were allocated only during heavy traffic (such as web browsing), and normal 3G channels were used when sending only keep-alives.

D. Wireless LAN

Wireless LAN measurements were performed in a Nokia office 802.11b/g network used mostly for voice-over-IP. No attempt was made to control for Wireless LAN network settings known to have impact on energy consumption, such as beacon and DTIM periods, U-APSD, and ARP caching (see [19] for discussion).

Fig. 6 shows the average power at different keep-alive intervals. The energy consumed by keep-alive messages is significantly smaller than in 2G and 3G, so the limitations of the measurement setup and background noise are more visible. However, we can estimate that the cost of a single keep-alive message to be around 200…400 mJ.

The type of keep-alive exchange was not important for WLAN.

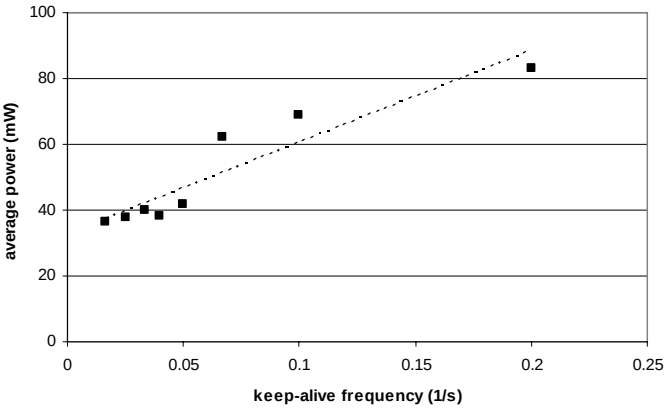

Fig. 6: Average Wireless LAN power consumption of keep-alive messages (with acknowledgment) with different intervals. The least-squares fitted line corresponds to 280 mJ/keep-alive plus 33 mW background consumption.

E. Discussion

Although we are not aware of other papers (beyond [8]) that measure energy consumption of keep-alive messages specifically, it is interesting to compare our results with other measurements about idle energy consumption.

Several authors have studied power consumption of IEEE 802.11 Wireless LANs. One recent paper by Agarwal et al. [2] reports that two CompactFlash 802.11b cards, intended for PDAs, consumed about 260 mW in idle (power save) mode. This figure includes only the network interface, not the PDA itself. The same paper also reports that Cingular 2125 smart

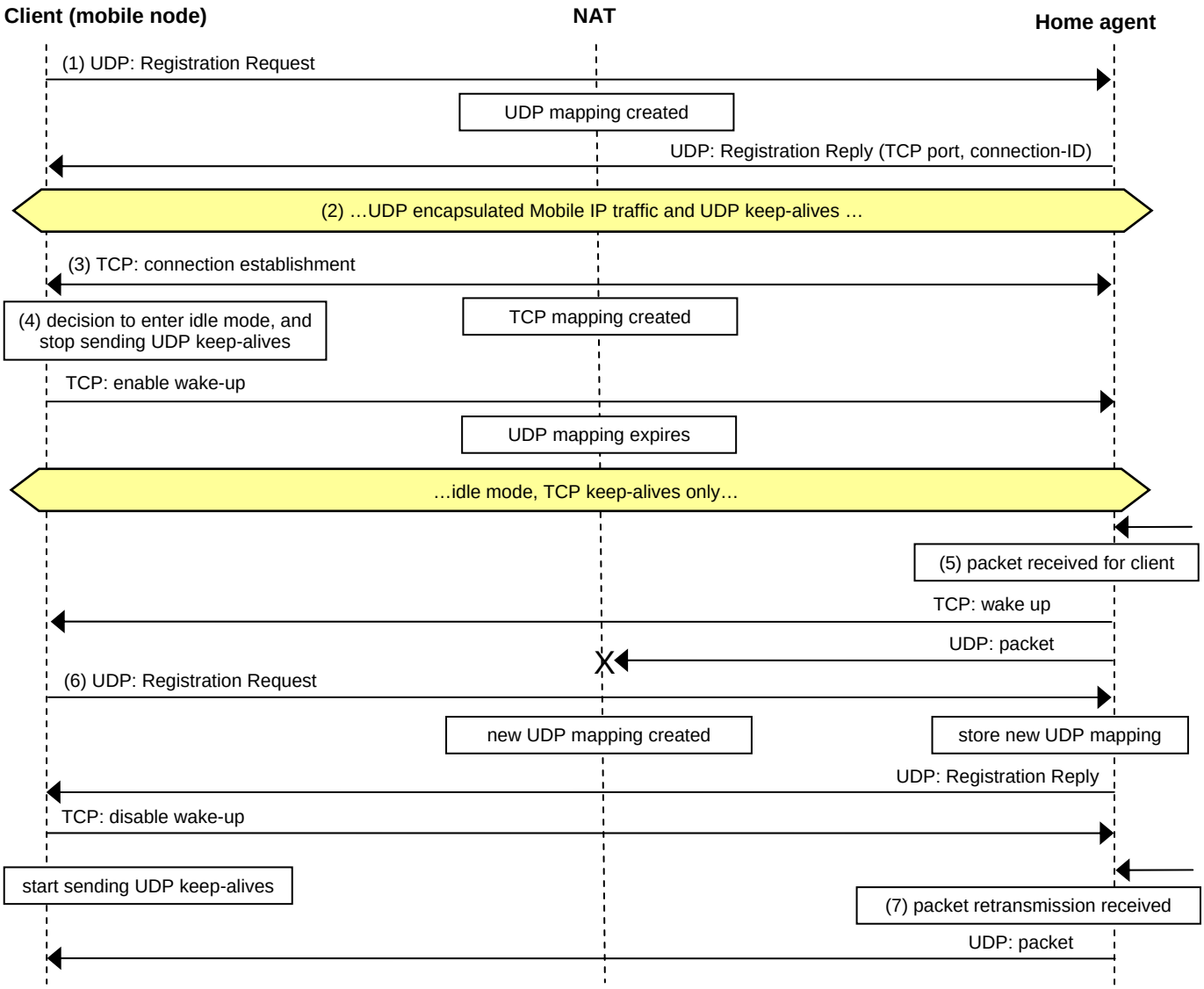

Fig. 7: Overview of TCP Wake-Up with Mobile IPv4.

phone (based on Windows Mobile 5.0) consumed 440 mW when connected to Wi-Fi network (this includes the whole phone). In another recent paper, Rahmati and Zhong report idle (power save) mode power consumption of 70…300 mW (for the wireless LAN interface alone) for three Windows Mobile-based smartphones [22].

These figures are remarkably different from our Wireless LAN measurements (33 mW background consumption for the whole phone), suggesting that low-power Wireless LAN chipsets are an area of active development, and vary significantly from product to product.

III. TCP WAKE-UP

In this section, we introduce "TCP Wake-Up", an extension to Mobile IPv4 and IPsec NAT traversal mechanisms.

A typical session using this extension would look as follows (also shown in Fig. 7):

1. During the Mobile IPv4 registration, the home agent provides the client with the information it needs to establish a TCP connection (a port number) and link it to the Mobile IPv4 registration (a connection identifier). A secret key that will be used to authenticate the TCP connection is also established. The extensions to Mobile IPv4 registration request/reply messages are described in detail in Section III.A, and the corresponding IPsec/IKE extensions in Section III.B.

2. When the client is not idle, it sends UDP keep-alives as usual.

3. The client establishes a TCP connection to the port given by the home agent, sends the connection identifier, and performs authentication using the key established earlier. The TCP connection can be established when the client registers with the home agent, or it can be postponed until the client is about to enter idle mode. The TCP-based protocol is described in detail in Section III.C, and the authentication mechanism (and its justification) in Section III.D.

4. When the client has become idle, it informs the home agent (requesting the home agent to send TCP-based wake-ups), and stops sending normal UDP keep-alives. The client detects that it has become idle based on, e.g., the amount of recently sent and received packets in the Mobile IPv4 tunnel.

After a while, the UDP NAT mapping expires. Keep-alives are still sent over the TCP connection to keep its mapping alive.

5. Later, when the home agent has a packet to be sent to the client, it sends a wake-up message over the TCP connection. (As before, it also sends the packet using UDP encapsulation, although it is quite likely this packet will be dropped by the NAT since it cannot find the correct mapping.)

6. When the client receives the wake-up message, it sends a new registration request message. This causes a new NAT mapping to be created, and communicates this information to the home agent (most likely the port number assigned by the NAT has changed). The client also starts sending UDP keep-alives again, and informs the home agent that it has left idle mode (TCP-based wake-ups are no longer needed).

7. Eventually, the correspondent node that sent the packet in step 5 will retransmit; as the NAT mappings are now fully up-to-date, the packet will be received as usual. (A possible optimization is to buffer the packet at home agent, and send it only once the new NAT mapping is ready; this is discussed in Section IV.D.)

Waking up can also be triggered by an outgoing packet at the client. The procedure for updating the NAT mapping and leaving idle mode is the same as above.

The procedure for IPsec VPNs is essentially similar, except that the information in step 1 is carried in IKEv2 messages, and in step 6, any authenticated message is sufficient to update the NAT mapping information in the VPN gateway.

A. Mobile IPv4 Extension

TCP Wake-Up requires an extension to the Mobile IPv4 registration messages. The basic requirements are to (1) determine whether the home agent supports this feature, (2) to communicate the TCP port number and (3) a connection identifier, and (4) to agree on a key used to secure the TCP connection.

To take the TCP Wake-Up feature in use, the client includes a TCP-WAKE-UP-SUPPORTED extension in the registration request. If the home agent supports this feature, the home agent includes a TCP-WAKE-UP extension in the registration reply. If the home agent does not support the feature, it ignores the client's extension and responds as usual.

The TCP port number and connection identifier could be either agreed implicitly (e.g., use a well-known TCP port, and the home address as the connection identifier), or explicitly communicated. To provide implementation flexibility, we decided to let the home agent select these, and included them the TCP-WAKE-UP extension in the registration reply.

The final requirement is to agree on a key to be used to authenticate the TCP connection. The client and home agent already share a key, the Mobile Node–Home Agent (MN–HA) key, which is used to authenticate the registration messages. We decided to derive a new key from this existing key. Since the MN–HA authenticator field is usually calculated with HMAC-MD5, we derived a new key with as HMAC-MD5(MN–HA key, 0x00 | "TCP Wake-Up" | nonce). The leading zero byte guarantees that we do not conflict with the use of MN–HA key for computing the MN–HA authenticator field, and a nonce selected by the home agent provides freshness.

B. IKEv2 Extension

In case of IPsec, similar extensions can be added to IKEv2 [13] messages. To indicate support for TCP Wake-Up, the client includes a TCP_WAKE_UP_SUPPORTED notification in the IKE_AUTH request. If the gateway supports TCP Wake-Up, it replies with a TCP_WAKE_UP notification, which contains the TCP port number and connection identifier.

These IKE messages are encrypted, so the key for TCP Wake-Up is randomly generated by the gateway, and sent to the client.

C. TCP Wake-Up Protocol

The TCP Wake-Up protocol is very simple, and there are only three different message exchanges. The same protocol can be used with both Mobile IPv4 and IPsec.

The "Start" message is sent by the client once it has established the TCP connection, and contains the connection identifier agreed on during Mobile IPv4 registration or IKE SA establishment. The home agent/gateway uses the connection identifier to find its local state, and replies with a "Challenge" message, containing a random nonce used to authenticate the client. The client completes the exchange with "Response" message, containing a MAC calculated using the agreed-on key and the challenge provided by the home agent.

The second possible exchange is used to enable or disable TCP-based wake-ups; the client sends either "Enable" or "Disable" message (consisting of a single byte), and the home agent/gateway replies with an "Ack". This exchange can also be used to verify that the TCP connection is working.

The third exchange is the actual "Wake-Up" message; a single byte sent by the home agent/gateway, which acknowledged by the client.

D. Security Mechanisms

The authentication of the TCP connection between the client and home agent/gateway deserves some explanation. In particular, why is authentication needed at all, and why the individual messages are not protected as well?

There are basically three different threats associated with the TCP wake-up connection.

First, an attacker could open a TCP connection to the home agent and pretend to be a valid client. The attacker would then receive notifications when the client has incoming packets. This would allow an attacker who is not otherwise able to eavesdrop the packets to perform some kind of traffic analysis. This threat is mitigated by requiring that the parties are authenticated when the TCP connection is established, and the key is agreed in a secure way.

Second, an attacker could prevent the client from waking up when it should, causing incoming packets to be dropped by the NAT. This attack can be carried out by attackers who are on the path between the client and the gateway; cryptographically protecting the wake-up messages would not change the situation.

Third, an attacker could unnecessarily wake up the client without a good reason, leading to unnecessary power consumption (called "sleep deprivation torture" by Stajano and Anderson [29]). This attack can also be carried out by an attacker who is on the path between the client and the

gateway, or is otherwise able to send packets that reach the client. In general, cryptographically protecting the wake-up messages would not change the situation significantly, since the client has already woken up to verify the packet.

To summarize, the authentication of the TCP connection is intended to counter the first threat, traffic analysis by off-path attackers. The two latter threats would not be significantly affected by per-message cryptographic protection, so that was not done in order to keep the protocol as simple as possible.

It could be claimed that the first threat is also quite unrealistic, and could be mitigated by requiring that the gateway chooses connection identifiers in an unpredictable manner. This is to some degree true, but it was felt the protocol looks more elegant this way.

IV. Analysis

In this section, we analyze the impact of TCP Wake-Up.

Subsection A describes the keep-alive message intervals used for this analysis, and their justification. Based on the selected keep-alive interval, Subsection B estimates how much energy could be saved, and Subsection C performs similar calculations for traffic volume.

The savings in energy consumption and traffic volume do not come without some costs: Subsection D discusses additional delay due to TCP-based wake-up, and Subsection E describes implementation considerations.

A. Keep-Alive Message Interval

Table 1, reproduced from [8], shows the default connection state timeout values for some common NAT and firewall products.

For the reminder of this section, we assume that UDP-based traffic will require keep-alive messages once every 30 seconds, and TCP will require 600 seconds (10 minutes). This is a somewhat pessimistic assumption, as some products use significantly longer timer values. For example, NATs compliant with the IETF BEHAVE working group specifications use a default timeout of at least 124 minutes for TCP and 120 seconds for UDP [3][7].

However, reliably determining what timeout is used by the NAT is difficult. For example, earlier versions of STUN specification [23] included a "binding lifetime discovery" procedure; however, this was removed from the main STUN specification, as it was found to be brittle and prone to error [24]. Mobile IPv4 and MOBIKE [6] messages can, in some cases, be used to detect NAT timeouts.

Most importantly, these lifetime discovery mechanisms usually work on with NATs, not stateful firewalls.

Product	TCP timeout	UDP timeout
Check Point NG FP2 firewall	60 min	40 s
Cisco IOS router NAT	1440 min	300 s
Cisco PIX firewall	60 min	120 s
Juniper Netscreen firewall	30 min	60 s
Nokia IP VPN gateway	60 min	120 s
ZyXEL Prestige 660W/HW ADSL router	60 min	60 s
ZyXEL ZyWALL 70 firewall	150 min	180 s

Table 1: Default connection state timeout values for some firewall/NAT products (reproduced from [8])

B. Energy Consumed by Keep-Alive Messages

Fig. 8 shows the estimated average power for normal Mobile IPv4 NAT traversal (with 30-second UDP keep-alives) and TCP Wake-Up (with 600-second TCP keep-alives), for 2G, 3G, and WLAN.

The figure shows that with the RNC configuration measured in Section II, TCP Wake-Up would have *four times* longer stand-by time in 2G, and over *seven times* longer stand-by time in 3G. For WLAN, the effect would be minimal.

It should be noted that, as shown in [8], the numbers depend heavily on RNC configuration. However, even with the most optimistic RNC configuration described in [8] (CELL_PCH enabled, T2 set to 2 seconds), TCP Wake-Up would still double the stand-by time. This is clearly a huge improvement.

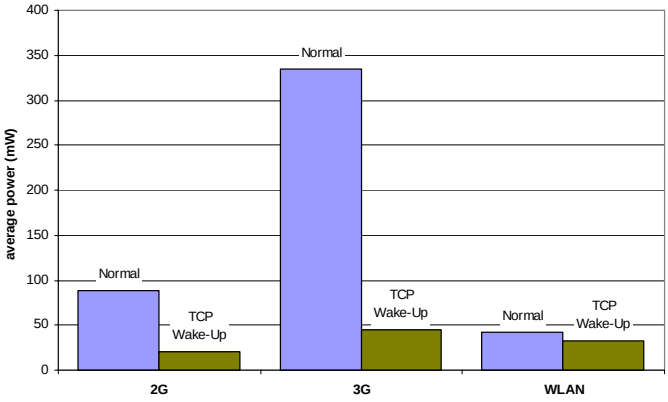

Fig. 8: Estimated power consumed by normal Mobile IPv4 NAT traversal (30-second UDP keep-alives) vs. TCP Wake-Up (600-second TCP keep-alives).

C. Keep-Alive Message Traffic Volume

While the most important cost of keep-alive messages is probably energy consumption, they also consume bandwidth.

In the basic Mobile IPv4 case, a keep-alive message consists of the following fields (the reply packet sent by the home agent is of the same size):

Field(s)	Size (bytes)
Outer IPv4 header	20
UDP header	8
MIP tunnel data message header	4
Inner IPv4 header	20
ICMPv4 header	8
Echo data	0+
Total	**60+**

For IPsec, the packet is as follows (the gateway does not reply to the keep-alive message):

Field(s)	Size (bytes)
Outer IPv4 header	20
UDP header	8
Keep-alive payload	1
Total	**29**

For TCP Wake-Up (reply packet is similar size; the final ACK from client does not have any data)

Field(s)	Size (bytes)
Outer IPv4 header	20
TCP header	20
TCP options and padding	0+
Data	1
Total	**41+**

Fig. 9 shows the total keep-alive traffic volume for a home agent/gateway with 10,000 clients. For Mobile IPv4, TCP Wake-Up reduces the daily traffic volume from 3.5 GB to 180 MB. However, while the decrease—over 3 GB—sounds large, it is probably small compared to the total traffic volume of a home agent with 10,000 clients.

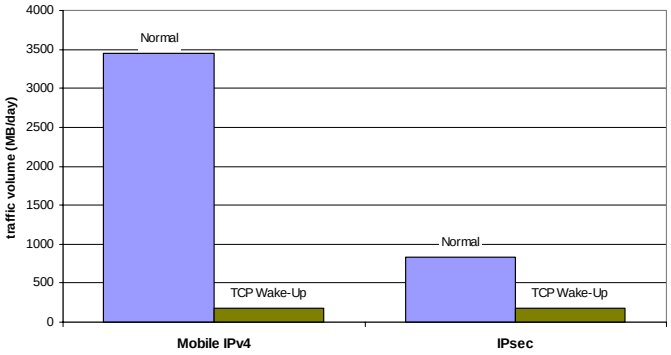

Fig. 9: Keep-alive traffic volume (per day) for a home agent/gateway with 10,000 clients.

D. Wake-Up Delay

In the design considered so far, the home agent/gateway never buffers data packets directed to the client. This means that the first data packet after wake-up is usually dropped by the NAT, since the necessary state has not been established yet. This is not very serious, since the host trying to contact the client has to accommodate for packet loss anyway, and will eventually retransmit.

However, waiting for the host to retransmit does result in some extra delay. The significance of this delay depends on the applications used. For example, email end-to-end delivery time is at least tens of seconds, so small additional delay is inconsequential. For voice-over-IP call establishment, the delay is directly visible to the calling party, so optimizing it is more important.

A possible optimization would be for the gateway to buffer the packet, and transmit it once the new NAT mapping has been established. This would reduce the extra delay due to TCP Wake-Up to roughly 1 round-trip time (assuming that the delay is mostly due to round-trip time, and not, e.g., processing time, is probably reasonable in 2G/3G). However, this would increase implementation complexity of the IP forwarding path.

E. Implementation Considerations

TCP Wake-Up introduces at least one potential scalability concern for home agent/gateway implementations. Historically, TCP/IP stacks were not designed to handle a very large number of concurrent TCP connections (see e.g. [14]). With TCP Wake-Up, a large home agent or VPN gateway could potentially require tens of thousands of TCP connections.

This question has been investigated by Shemyak and Vehmanen [25] in the context of SIP servers; their results show that scalability depends significantly on implementation details, but when proper APIs are used, even a low-end Linux box can handle 100,000 simultaneous TCP connections.

However, using TCP may still complicate implementation of high-availability features, such as transparent synchronization with a "stand-by" backup node.

V. Alternative Solutions

In some sense, TCP Wake-Up is a "patch solution" that would not be needed in an ideal networking system. This section discusses alternative solutions to the problem of keep-alive energy consumption.

The obvious alternatives, reducing the energy consumed by a single keep-alive message, and using longer NAT/firewall timeouts, are discussed in Subsections A and B, respectively. Subsection C describes the use of TCP-based protocols in general, and Subsection D examines approaches that avoid the need for keep-alives completely by sending wake-up messages "out-of-band" (not going through the NAT/firewall).

A. Reducing the Cost of Single Keep-Alive

The results of Section II and [8] show that the energy consumption of a single keep-alive exchange depends heavily on the details of the particular radio technology and its parameters. While energy consumption has been considered in the design of these radio technologies, it has focused largely on idle mode (with no traffic at all) and voice calls, and regular "background traffic", such as keep-alives, has received little attention.

Some HSDPA features, such as HS-SCCH-less operation and enhanced CELL_FACH (see Peisa et al. [20]) could be expected to eventually decrease keep-alive energy consumption. These features decrease signaling overhead and latency, and thus enable using shorter RRC timer values without sacrificing user experience. However, the measurements in Section II show that these savings are not necessarily realized in current HSDPA networks.

Energy consumption of Wireless LANs has also received significant attention from researchers (see e.g. [21] and its references); however, a more detailed discussion is beyond the scope of this paper.

B. Increase Keep-alive Interval

Haverinen et al. [8] and the IETF BEHAVE working group specifications [3][7] recommend using relatively long timeouts in NATs and firewalls, reducing the need for keep-alive messages. While such recommendations can improve the situation in the long term, in short term the NATs are often beyond the control of, e.g., consumers and enterprises wanting to use always-on applications with mobile phones. Also, as noted in Section IV.A, reliably determining the timeout currently in use is not simple; recent work in this area includes

the Self-Address Fixing Evolution (SAFE) proposal in IETF [12].

Middlebox signaling protocols (see [5] for one recent survey) would allow the client to explicitly request a longer timeout for a particular traffic flow, but such protocols have not seen widespread deployment, especially in 2G/3G networks.

C. Use TCP-based Protocols

One obvious solution for frequent UDP keep-alives is, of course, to use TCP instead. This could mean, for example, handling connection disruptions (such as IP address changes due to mobility) in the application layer—which many applications already do—and using SSL/TLS for security, either end-to-end or with SSL VPNs.

It is also good to note that some IPsec products do support TCP encapsulation for IPsec (e.g., "Visitor mode" in Check Point SecureClient [4]), and that some SSL VPN products, despite their name, actually do not use SSL or TCP for all traffic [27]. Thus, the performance differences between these approaches are not necessarily obvious; for example, Snyder [27] discovered that running voice-over-IP traffic over TCP does not necessarily degrade call quality.

D. Out-of-Band Wake-Up

A device with multiple network interfaces can also save energy by completely powering off some interfaces, and possibly leverage energy consumption differences between radio technologies. For example, Shih et al. [26] propose including a secondary low-power radio interface, which is used to wake up the client, allowing switching off the main Wireless LAN interface. Agarwal et al. [1] propose a similar scheme using Bluetooth as the secondary radio.

Other proposals use non-TCP/IP based protocols to completely remove the need for NAT/firewall keep-alive messages. For example, a number of protocols defined by Open Mobile Alliance (OMA), such Multimedia Messaging Service (MMS) and Device Management (DM), use text messages (SMS) to "wake up" the phone. Agarwal et al. [2] GSM caller ID signaling as the wake-up channel. However, using non-TCP/IP-based protocols complicates the system, and may require operator involvement.

VI. Conclusions and Future Work

In this paper, we have analyzed the energy consumption of keep-alive messages on a mobile phone with 2G, 3G, HSDPA and Wireless LAN networks. The results show that "always-on" applications with long-lived persistent connections cannot be based solely on UDP if they aim to be usable in current 2G/3G networks.

To allow Mobile IPv4 and IPsec VPNs to be used in such environments, we introduce TCP Wake-Up, an extension to Mobile IPv4 and IPsec NAT traversal mechanisms which avoids the need for UDP keep-alives when the connection is idle. The analysis in Section IV shows that TCP Wake-Up could extend the battery lifetime by a factor of 2 to 7.

TCP Wake-Up requires changes only in the client and home agent/VPN gateway, but not in the network elements (such as radio access networks, NATs, and firewalls) between them, and thus it can be deployed by e.g. an enterprise. This differentiates TCP Wake-Up from many other solutions discussed in Section V.

However, improvements in other areas—such as HSDPA energy consumption with improved signaling procedures—could eventually reduce the improvement offered by TCP Wake-Up. Future studies on these topics are therefore recommended. Future work could also include extending the TCP Wake-Up concept to other protocols that include IP-over-UDP tunneling, such as Dual-Stack Mobile IPv6 [28] and Teredo [10].

Acknowledgment

The authors would like to thank N. Asokan, Dan Forsberg, and Henry Haverinen for their valuable comments and suggestions, and Antti Miettinen and Gerard Bosch Creus for their help with the energy consumption measurements.

After we had completed the TCP Wake-Up specification, we discovered that something similar—using a "back-up" TCP connection between IPsec VPN client and gateway—had been proposed by David Mason already in 2001 [16]. However, as far as we know, this proposal never went beyond one email on the IETF IPsec working group mailing list.

References

[1] Y. Agarwal, C. Schurgers, and R. Gupta, "Dynamic Power Management Using On Demand Paging for Networked Embedded Systems", Proc. Asia South Pacific Design Automation Conference, 2005.

[2] Y. Agarwal et al., "Wireless Wakeups Revisited: Energy Management for VoIP of Wi-Fi Smartphones", Proc. ACM MobiSys, 2007.

[3] F. Audet and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", RFC 4787, 2007.

[4] Check Point, "VPN-1 SecureClient Datasheet", 2007.

[5] L. Eggert et al., "A Survey of Protocols to Control Network Address Translators and Firewalls", work in progress (draft-eggert-middlebox-control-survey-01), 2007.

[6] P. Eronen, "IKEv2 Mobility and Multihoming Protocol (MOBIKE)", RFC 4555, 2006.

[7] S. Guha et al., "NAT Behavioral Requirements for TCP", work in progress (draft-ietf-behave-tcp-07), 2007.

[8] H. Haverinen, J. Siren, and P. Eronen, "Energy Consumption of Always-On Applications in WCDMA Networks", Proc. IEEE VTC 2007-Spring.

[9] O. Honda et al., "Understanding TCP over TCP: effects of TCP tunneling on end-to-end throughput and latency", Proc. SPIE –Volume 6011 Performance, Quality of Service, and Control of Next-Generation Communication and Sensor Networks III, M. Atiquzzaman, S.I. Balandin (eds.), 2005.

[10] C. Huitema, "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)", RFC 4380, 2006.

[11] A. Huttunen et al., "UDP Encapsulation of IPsec ESP Packets", RFC 3948, 2005

[12] IETF, "Proceedings of the Seventieth Internet Engineering Task Force, Self-Address Fixing Evolution (safe)", http://www3.ietf.org/proceedings/07dec/safe.html, 2007.

[13] C. Kaufman, "Internet Key Exchange (IKEv2) Protocol", RFC 4306, 2005.

[14] D. Kegel, "The C10K problem", http://www.kegel.com/c10k.html, 2006.

[15] H. Levkowetz and S. Vaarala, "Mobile IP Traversal of Network Address Translation (NAT) Devices", RFC 3519, 2003.

[16] D. Mason, "RE: I-D ACTION:draft-ietf-ipsec-udp-encaps-00.txt", message on ipsec@lists.tislabs.com mailing list, July 12, 2001, http://www.sandelman.ottawa.on.ca/ipsec/2001/07/msg00066.html

[17] Nokia, "Device Details: Nokia E65", http://www.forum.nokia.com/devices/E65, 2007.

[18] Nokia, "Device Details: Nokia N95", http://www.forum.nokia.com/devices/N95, 2007.

[19] Nokia, “Recommendations for Reducing Power Consumption of Always-On Applications”, http://www.forum.nokia.com/main/resources/development_process/power_management/, 2007.
[20] J. Peisa et al., “High Speed Packet Access Evolution – Concept and Technologies”, Proc. IEEE VTC 2007-Spring.
[21] X. Pérez-Costa, D. Camps-Mura, and A. Vidala, “On distributed power saving mechanisms of wireless LANs 802.11e U-APSD vs 802.11 power save mode", Computer Networks, 51(9):2326–2344, 2007.
[22] A. Rahmati and L. Zhong, “Context-for-Wireless: Context-Sensitive Energy-Efficient Wireless Data Transfer”, Proc. ACM MobiSys, 2007.
[23] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy, “STUN – Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)”, RFC 3489, 2003.
[24] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, “Session Traversal Utilities for (NAT) (STUN)”, work in progress (draft-ietf-behave-rfc3489bis-13), 2007.
[25] K. Shemyak and K. Vehmanen, “Scalability of TCP Servers Handling Persistent Connections”, Proc. 6th International Conference on Networking, 2007.
[26] E. Shih, P. Bahl, and M.J. Sinclair, “Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices”, Proc. ACM MobiCom, 2002.
[27] J. Snyder, “Test shows VoIP call quality can improve with SSL VPN links”, Network World, http://www.networkworld.com/reviews/2006/022006-ssl-voip-test.html, 2006.
[28] H. Soliman, “Mobile IPv6 support for dual stack Hosts and Routers (DSMIPv6)”, work in progress (draft-ietf-mip6-nemo-v4traversal-06), 2007.
[29] F. Stajano and R. Anderson, “The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks”, 7th International Workshop on Security Protocols, LNCS vol. 1796, 1999.
[30] O. Titz, “Why TCP Over TCP Is A Bad Idea”, http://sites.inka.de/~W1011/devel/tcp-tcp.html, 2001.

