Wave & Wait Protocol (WWP): High Throughput & Low Energy for Mobile IP-Devices.

V. Tsaoussidis, A. Llahanas
College of Computer Science
Northeastern University, Boston, MA 02115

H. Badr
Dept. of Computer Science
SUNY at Stony Brook, NY 11794

Abstract

The work we present here reports on the further development and testing of an experimental transport-level protocol. The protocol runs on top of IP, and is intended for mobile, wireless devices. Its central concern is to conserve battery-powered energy used for transmission while maintaining high levels of data throughput. It can be adjusted to achieve higher throughput at the expense of lesser energy-saving in accordance with the application's needs. It conserves transmission effort, adjusting data transmission below perceived network congestion level, and thereby minimizes the need for duplicate data retransmission due to congested routers losing packets, and so on. It achieves high throughput by implementing a network probing mechanism that enables it to investigate instances in which prevailing throughput conditions appear to be deteriorating, before committing to further data transmission. The probing mechanism also permits it to continuously monitor the network during high congestion periods in order to locate windows of sufficiently low congestion opportunity to exploit for transmission.

1. Introduction

Mobile communication devices are increasingly dominating today's market. Many applications are being developed for mobile stations, which require very specific protocol support depending on the characteristics of both application (*e.g.*, e-mail, web, multimedia) and mobile station (*e.g.*, laptop, phone, handheld etc.). For the environments in which such stations operate, throughput is not the sole dominant performance criterion. Energy-saving becomes a crucial factor as well.

Most recent studies ignore the energy/throughput tradeoff at the transport layer, although it is becoming a key performance issue. The only published studies of TCP energy consumption are presented in [4, 6]. The authors in [6] present results, based on a stochastic model of TCP behavior. While the model makes some assumptions in order to maintain analytic tractability, it remains unclear how these assumptions might invalidate the results. The authors' conclusions are based on the additional energy expenditure caused by channel impairments only in the context of retransmitted data; experimental results based on the combination of time and overhead are presented in [6]

The protocol presented here runs on top of IP, and is intended for mobile, wireless devices. Its central concern is to conserve battery-powered energy used for transmission while maintaining high levels of data throughput. It can be adjusted to achieve higher throughput at the expense of lesser energy-saving in accordance with the application's needs. It conserves transmission effort, adjusting data transmission below perceived network congestion level, and thereby minimizes the need for duplicate data retransmission due to congested routers losing packets, and so on. It achieves high throughput by implementing a network probing mechanism that enables it to investigate instances in which prevailing throughput conditions appear to be deteriorating, before committing to further data transmission. The probing mechanism also permits it to continuously monitor the network during high congestion periods in order to detect and exploit windows of opportunity of improved conditions during which aggressive transmission of data can be successfully undertaken as conditions alter. A way to save energy is to avoid retransmissions, unnecessary headers, and redundant data. The less time expended on transmissions, the better the energy saving. For example, transmission of packets over a congested network will cause packets to be dropped and, consequently, a reliable protocol will initiate a retransmission mechanism. Instead, our protocol first "probes" the network to estimate prevailing levels of congestion risk and adjusts its transmission accordingly. It transmits aggressively when existing conditions appear to be favorable, and backs off as congestion is detected, thereby attempting to take maximum advantage of favorable conditions without wasting energy on transmissions that are unlikely to be successful as conditions deteriorate.

At the core of the protocol's throughput and energy-saving performance is its ability to monitor network conditions and rapidly adjust its transmission strategy as these conditions continuously change for the better or worse. Two mechanisms are central to providing the protocol with this capability. The transmission of a wave is used to monitor current network performance, and the next wave level is set in conformity with what is thereby detected about

0-7695-0777-8/00 $10.00 © 2000 IEEE

prevailing conditions. Secondly, probe cycles constitute an "energy-conserving" mechanism for investigating instances in which good prevailing conditions appear to be deteriorating, before committing to further data transmission, and to continuously monitor the network during high congestion periods in order to detect and exploit windows of opportunity of improved conditions during which aggressive transmission of data can be successfully undertaken as conditions alter. Probe cycles are "energy conserving" in contrast to the energy that would otherwise be expended on the transmission of data segments that do not have a good chance of getting through during high congestion periods. Because of the centrality of waves and probing to the protocol's innovative approach, we proceed to present each of these in turn in some detail. Details on other aspects of the protocol (*e.g.*, connection establishment, segment formats and types, *etc.*), can be found in [3].

2.2 Wave Mechanism

The protocol first groups data segments into waves on the sending side and then transmits the segments of a wave one after the other, rather than simply sending separate segments individually when it can. The reason is that, in order for the receiver to effectively estimate network congestion based on the successive segments reaching it, it needs some knowledge about the sender's pattern of transmission of these segments. This knowledge is (implicitly) provided by the fact that waves at a given level are made up of a predetermined, fixed number of data segments of fixed size, and the sender transmits the segments of the wave one after the other with no pause between one segment and the next.

While data segments are of fixed size, in any given implementation of the protocol the segment size can be set so as to optimize the average number of bytes that need retransmission, in line with the network's overall characteristics of burst errors, and so on. Similarly, the number of wave levels, and the fixed number of segments comprising each wave level, can also be set with an eye to the application's message sizes, as well as the protocol's own internal need for wave "granularity" matching the network's range of congestion behavior (*i.e.* small waves containing few segments for transmission under significant congestion, through to large waves containing many segments in order to exploit opportunities when congestion is low).

The receiver attempts to estimate prevailing congestion conditions by monitoring the throughput of the current wave and setting the level of the next wave accordingly. A wave at level i ($i>=0$) is composed of a fixed number $W(i)$ of data segments. For $i=0$, $W(0)$ is defined to be 0.

A data segment is composed of a 6-byte header and a fixed-sized data payload [3]. Once the first segment to reach the receiver from a new wave arrives, it is easy for the receiver, given the current wave level i (which is carried in the segment header), to calculate how long it would take the rest of the wave to reach it if the network were relatively uncongested, using a "*baseline*" throughput of *BT* KBytes per second for the uncongested network. The time thus calculated is the "*baseline time*". The receiver measures how long it actually takes for the remaining segments in the wave to arrive. It then uses the baseline and measured times for the wave to set the level of the next wave.

Our design currently calls for four wave levels, $i=0, 1, 2, 3$. The number of segments in a wave

$$W(i) = (12 \times i) \qquad for\ i=0, 1, 2, 3;$$

of fixed-sized segment payload was set to 1KByte. The following simple algorithm was used by the receiver to set the next wave level:

Suppose the current wave is at level i, $i = 1, 2, 3$.

```
Let T(i) be the measured time for a level
i wave.
Let B(i) be the baseline time for a level
i wave.
For j = 1, 2, 3:
if T(i) is in the range (j-1 , j] X B(i),
then
next wave level is set to (4-j);
else
set wave-level to 0.
```

The algorithm above essentially implies that the when the receiver sets the new wave level to k, $k = 1, 2, 3$, it is estimating the current network throughput to be no worse than approximately a fraction $1/(4-k)$ of the baseline throughput value of *BT* KBytes per second (and no better than approximately a fraction $1/(4-k)$, for $k = 1$ *or* 2). The number of segments in the new wave is then adjusted proportionately. If the throughput appears to be less than 1/3 of the baseline throughput, we go to level 0, deeming it better to pause for a while than risk expending energy transmitting even a small wave that might not have a sufficiently good chance of getting through undamaged. In the event the receiver sets the next wave level at 0, the sender will immediately probe the receiver. The receiver uses the RTTs (Round Trip Times) measured during the probe cycle to set the new wave level in the N-S_ACK segment (Negative Selective Acknowledgment) it sends to the sender at the end of the cycle (see Subsection 2.3 below). The sender does not start transmitting segments until it has a sufficient number to make up a complete wave. When it receives a N-S_ACK setting the wave level, and if it does not have sufficient old (needing retransmission) and new data up to the specified number of segments in the wave, it will transmit the segments it has at the highest wave level for which it has enough segments.

2.3 Probing Mechanism

The wave mechanism described above enables the receiver to estimate the current level of congestion and the associated delay for data segment delivery. This information is implicitly and indirectly summarized in the level selected for the next wave. However, an error that causes data segments from a wave to be lost might be an indication of either a purely transitory problem on the transmission line (such as a burst error, for example), or of deteriorating congestion conditions which are likely to be of longer duration. It should be noted before proceeding further that the probing mechanism described here is significantly different from that reported on in [3].

A probe cycle enables the receiver to measure two successive RTTs from the network, thereby providing it with sufficient information to determine whether the error was purely transitory or an indication of longer-lasting congestion build up. Operating at even high wave levels, and pausing to check with probes in the event of unduly delayed, and possibly lost, data segments, the receiver can decide whether to have data transmissions continue at an appropriately high wave level, adjust the wave level downwards, or even temporarily stop data transmission altogether. The approach takes full advantage of the fact that almost-current congestion conditions have already been estimated by the receiver, and are summarized by the current wave level. The receiver uses the absolute values of the two RTTs measured from the probe cycle, as well as the difference between these two RTTs, to set the next wave level in terms of an incremental change from the current level. The new wave level is signaled to the sender by means of a N-S_ACK segment that terminates the probe cycle.

A probe cycle starts when the N-S_ACK segment for a data wave just transmitted goes absent (see figure 1). This could indicate incomplete delivery of (or undue delay of some segments from) the wave, or the delay/loss of the wave's corresponding N-S_ACK. The SEND_T timer at the sender side expires and a PROBE1 segment is transmitted. The receiver responds with a PR1_ACK, upon receipt of which the sender transmits a PROBE2. The receiver acknowledges this second probe with a PR2_ACK and enters a state where it waits for a PROBE3. It makes an RTT measurement based on the time delay between sending the PR1_ACK and receiving the PROBE2. Upon receipt of PROBE3 it makes the second RTT measurement based on the time delay between the PR2_ACK and the PROBE3. The receiver then determines the level for the next wave and informs the sender by means of a N-S_ACK. In setting the next wave level, the receiver takes into account the network conditions that had been detected at the time the current wave level was determined, as well as the values of the two RTT measurements and the delay variation (jitter) between them. Other applications with bursty flows that are currently sharing the same links in the network that our application is being routed along will induce measurable jitter between the two RTTs. Contrariwise, an error free environment, or links that are being shared with normalized/smoothed-out data flows, will induce no jitter. The receiver can take all this into consideration in setting the level for the next wave. The full set of decision-making rules has not yet been completely standardized, but a set of rules has been developed, implemented, calibrated and used for our testing environment.

Although probing is a fairly complicated mechanism and adds additional RTTs to the protocol's progress, it proves to be a more useful device than would be sending data that is likely to be dropped, on the one hand; or reducing the window size (*i.e.*, reducing the wave level) and degrading the connection throughput, possibly for no good reason, on the other. The first option would negatively impact energy expenditure. The second would needlessly degrade the effective throughput and also, by unnecessarily prolonging the connection time, also impact energy consumption.

3. Implementation & Testing

The protocol was implemented using the x-kernel protocol framework [5]. The high-level test protocol sends messages of 1024 bytes to the underlying WWP layer. These are then buffered until there are enough segments to form a wave at the level needed. Congestion was simulated by dropping and delaying segments using modified x-kernel protocols. Our modified x-kernel protocol, which we call VDELDROP, drops segments at a constant rate specified for the duration of a test, and causes different delays for each segment. VDELDROP also has the capability of alternating On/Off phases during which its actions are in effect and are suspended, respectively. Thus, during a connection period, WWP would experience phases that are error free and others with simulated congestion error effects. This modification enabled us to test WWP's behavior in response to sudden changes in the simulated environment, and its ability to rapidly re-adapt to varying congestion conditions. Such conditions are typical of mobile networks where the user is "on the move": communication with the access points will have variable characteristics during the connection time. VDELDROP was configured above IP, with WWP configured on top of it, and our high-level testing ("application") protocol configured above WWP. We also ran TCP [2] under a similar configuration.

We compare our protocol with TCP Reno since TCP is a reliable protocol with end-to-end service similar to WWP. It is also a topic of current research interest with respect to its behavior in wireless environments. However, TCP does not distinguish well between congestion [1], on the one hand, and transient transmission burst errors, on the other, although each requires distinct actions in response to its occurrence (*e.g.*, slow down, and continue feeding the network, respectively).

The data segment payload was 1KByte; $W(i) = 12 \times i$, $i = 0,1,2,3$; All this probably causes WWP to understate its potential somewhat, though some effort was put into calibrating the sender timeout values SEND_T and PROBE_T, and the decision-making process by which the next wave level was determined at the end of a probe cycle, in order to enhance performance. The protocol clearly achieves higher throughput and conserves more energy than TCP. Furthermore, while both protocols' throughputs and energy-expenditures degrade as increasingly problematic network conditions are simulated, WWP's performance *relative* to TCP's improves, using their performance under error-free conditions as a reference point. This demonstrates the effectiveness of the protocol, irrespective of implementation or testing environment limitations that might not have been taken into account, and which could be constraining either protocol's absolute performance.

All tests are undertaken using 5-MByte (5,242,880 bytes) data sets for transmission. The purpose of the tests was to evaluate the behavior of the two protocols in response to changes in the simulated network environment, such as congestion and transmission errors at different rates and for different duration. We took measurements of the total connection time and of the total number of bytes transmitted (*i.e.*, including protocol control overhead transmissions, data segment retransmissions, *etc.*). Both factors significantly affect energy expenditure as well as throughput. Error conditions have two distinct characteristics – transmission errors, as opposed to excessive congestion - but one and the same result: segments are lost. Segments may or may not be delayed during the On phases of VDELDROP (the delay effect of VDELDROP is random). A challenge for both protocols - also tested for - was whether, in the presence of errors, the response would be to reduce the window size (*i.e.*, reduce the wave level in the case of WWP), or proceed aggressively with data transmission instead. Note that when burst errors, which by their nature are transient, occur and the sender reduces its window size in response, the achieved throughput will be below the maximum attainable under these conditions.

In Table 1 (appendix) we present results for the 5-MByte data messages and VDELDROP On/Off phase duration of 10 seconds. In order to represent the energy expenditure overhead required to complete reliable transmission under different conditions, we use the `Byte Overhead` as a metric. This is the total *extra* number of bytes the protocol transmits, over and above the 5 MBytes delivered to the application at the receiver, from connection initiation through to connection termination. The `Byte Overhead` is thus given by the formula:

`Byte Overhead = Total - Base,` where,

`Base` is the number of bytes delivered to the high-level protocol at the receiver. It is a fixed 5 MBytes for all tests.

`Total` is the total of all bytes transmitted by the transport layers , and is given in the column **Total Bytes**. This includes protocol control overhead, data segment retransmission, as well as the delivered data.

The time overhead required to complete reliable transmission under different conditions is given in column **Time overhead** using the formula: `Time Overhead = Connection Time - Base,` where,

`Base` is the number of seconds required to deliver the 5 MBytes to the high level protocol at the receiver under error-free conditions, from connection initiation through to connection termination (column **Time Ran** for the test set 1 in the table).

`Connection Time` is the corresponding amount of time required for completion of data delivery under error-prone conditions, and is given in column **Time Ran**.

The net performance of the protocols is given in column **Goodput** using the formula: `Goodput = Original Data / Connection Time,` where, `Original Data` is the 5-MBytes data set.

The **Drop Rate** reported is the dropping rate for segments during the On phases, not the averaged overall drop rate across On/Off phases.

As demonstrated by the results for test sets 2, 3, 4, 5, and 6, the behavior of WWP is very constant with respect to time (and hence throughput) and energy expenditure. The throughput achieved is far better than TCP's and the energy expenditure is far less. WWP adapts quickly to error phases. It does not automatically decrease its sender window size in response to a drop as does TCP. Instead, immediately upon experiencing a drop, it pauses in its data transmission and probes. It then checks the measured RTTs from the probe cycle to assess whether conditions allow continued transmission and, if so, at what level. It can adjust back immediately to a high wave level where appropriate, unlike TCP which applies graduated multiplicative/additive increases to its window size. This mechanism of WWP's has two significant results: (i) data transmission is not wasted during sustained periods of degraded network capacity, and thus retransmissions are reduced to a minimum; and (ii) throughput is maximized since we can adjust to high wave levels immediately under appropriate conditions, thereby not wasting opportunities for successful data transmission by attempting less than the network would easily be able to accommodate. This is clearly demonstrated by the results for test sets 5 and 6.

There, TCP reduces its window size significantly since the error rate is high. It then takes considerable amounts of time to readjust the size back up to an appropriate level, thereby "missing" the "good" phase. This results in trading off prolongation of the connection time in order to avoid retransmissions. The protocol is clearly inefficient under such conditions.

Table 2 (appendix) outlines the effectiveness of the protocols with respect to energy savings and throughput, as well as their relative behavior under varying conditions. Column **Overhead Ratio** is a measure of the relative transmission overhead of the two protocols, and is calculated as TCP (Total Bytes Transmitted - 5 MBytes) / WWP (Total Bytes Transmitted - 5 MBytes).

Goodput Ratio gives the relative throughput rates: `TCP Goodput / WWP Goodput`. Finally, **Time Ratio** shows the relative connection times for the two protocols (TCP/WWP).

When error rates are low, TCP behaves well and its major weakness - inability to adequately readjust to rapidly changing conditions – does not catastrophically degrade its performance. WWP displays consistently good behavior, even when variability in the error environment is quite dramatic. Under such conditions of high variability, TCP wastes enormous amounts of energy and unexploited throughput capacity, and does not achieve good performance. Its throughput in all error-prone cases is well below what it is capable of achieving under error-free conditions, let alone the higher throughput displayed by WWP. As demonstrated by test sets 3 and 4, it is unable to take advantage of network conditions, expending up to fourfold more time than WWP. It is interesting to note (see test set 3) that TCP transmits up to 158 KBytes more than does WWP under the same conditions, in order to deliver its 5 MByte data set. So it is not even trading off lesser transmission effort at the expense of longer connection times in an effective manner. Indeed, it is the longer connection times of TCP that seriously aggravate the energy consumption characteristics of the protocol, rather than wasted transmission effort as such. Throughput can reach as low as only 23% of that achieved by WWP, although under error-free conditions it can achieve up to 58.6% of WWP's corresponding goodput. Similarly, under error-free conditions, their relative time ratio was 1.7/1; this ultimately grows to about 4/1 as error conditions deteriorate.

Results presented in the table for the relative data overhead deteriorate to a value of 4.48/1, which portends that energy expenditure should reach even worse values were the total connection times to be taken fully into account.

4. Conclusion

We have presented a reliable transport protocol that achieves high throughput and considerable energy-saving in network environments with significantly variable error characteristics. Our results clearly demonstrate that retransmissions and duplications can be avoided if we transmit data when network conditions allow it. Comparison results at different congestion levels demonstrate suitable behavior by the protocol. In contrast to TCP, totals for overall bytes transmitted as well as communication time are significantly less, so energy saving is proportionally higher.

5. References

1. M. Allman, V. Paxson, W. Stevens, "TCP Congestion Control", RFC 2581, April 1999
2. J. Postel, Transmission Control Protocol, RFC 793, September 1981.
3. V. Tsaoussidis, H. Badr, R. Verma, "Wave and Wait Protocol: An energy saving transport protocol for mobile IP-Devices", IEEE ICNP '99, Toronto, Canada, October 1999.
4. V. Tsaoussidis, H. Badr, "Energy / Throughput Tradeoffs of TCP Error Control Strategy", IEEE Symposium on Computers and Communications, IEEE ISCC 2000, France, 2000.
5. The X-kernel: www.cs.arizona.edu/xkernel
6. M. Zorzi, R. Rao, "Energy Efficiency of TCP", MoMUC '99, San Diego, California, 1999

T #	Prtcl	Drop Rate	Time Ran	Total Bytes	Time Overh.	Byte Overh.	Goodput
1.1	TCP	0	69	5319120	---	76240	75983
1.2	WWP	0	40.5	5268495	---	25615	129453
2.1	TCP	5%	117	5395782	48.4	152902	44685
2.2	WWP	5%	71	5290519	30.5	47639	73843
3.1	TCP	10%	133	5409384	63.9	166504	39449
3.2	WWP	10%	81.9	5295859	41.4	52979	64015
4.1	TCP	20%	136	5410822	67.63	167942	38372
4.2	WWP	20%	84	5319556	43.58	76676	62355
5.1	TCP	33%	258	5392682	189.8	149802	20251
5.2	WWP	33%	85.9	5324601	45.4	81721	61034
6.1	TCP	50%	259	5388348	190.3	145468	20213
6.2	WWP	50%	86.7	5275294	46.2	32414	60471

Table 1: 10sec On/Off Drop phase, 5MB data transmission

T #	Prtcl	Drop Rate	Time Ran (sec)	Overhead Ratio	Goodput Ratio	Time Ratio
1.1	TCP	0	69	2.97/1	58.6%	1.70/1
1.2	WWP	0	40.5			
2.1	TCP	10%	132-135	2.62-3.14/1	59-60%	1.62-1.64/1
2.2	WWP	10%	81.3-81.9			
3.1	TCP	33%	258-368	1.83-2.56/1	23-32%	3.0-4.2/1
3.2	WWP	33%	85.9-87.6			
4.1	TCP	50%	259-385	2.0-4.48/1	24-33%	2.98-4.1/1
4.2	WWP	50%	86.7-93.7			

Table 2: Summary of results with Drop Phase 4-10 sec