(RCom) Reliable Communications for Teleoperated Rescue Robots

Ben Axelrod (baxelrod@cc.gatech.edu)
John Envarli (envarli@cc.gatech.edu)

Abstract

We have created a reliable wireless communication framework for use in highly congested environments, and where high throughput is desired. This is performance is achieved through the use of multiple orthogonal channels. Our system has been developed from the ground up specifically for the RoboCup Rescue competition.

We developed a simple Additive Increase and Decrease (AID) approach that reduces the speed by a constant for dropped packets, and increases the speed by a constant for delivered packets. We were able to achieve higher throughput with this approach than both using a fixed optimal speed and a multiplicative decrease method that is similar to TCP's implementation. We were also able to make significant improvements to the communication system, through implementing a sending queue, probabilistic selection of optimal links and retransmission of dropped packets.

1 Introduction

1.1 RoboCup Rescue

RoboCup Rescue is an annual international competition where teleoperated robots are deployed in a simulated disaster environment in order to locate victims. Challenges to the participants include mobility of the robot in the rough terrain, building a map to present to potential rescue workers, and solving the teleoperation problem. In this domain, teleoperation means wirelessly streaming enough video data back to the human driver to quickly locate victims and also localize his/herself in cluttered and dimly lit environments.

This project focused on the wireless communication aspect of teleoperation. This problem is non-trivial due to the nature of both wireless communications, and the RoboCup competition. During the actual RoboCup competition, several other leagues could potentially be taking place at the same time, all co-located in a rather small area. Almost every league utilizes wireless communication, congesting almost every frequency in the area.

Another inherent problem that needs to be addressed is the incompatibility of TCP and wireless. TCP assumes a packet drop is always the result of congestion, whereas in a wireless connection this is far from the truth: Random packet drops happen frequently without congestion due to interference or the inherent imperfection of a wireless connection [1]. TCP also ensures delivery of packets, which is not necessarily what one wants when trying to stream the latest possible frame of video data. We retransmit when possible and feasible but do not guarantee delivery of a data segment.

1.2 RCom & Multiple Links

Our hardware solution is to use three orthogonal wireless links (via wireless bridges). These links are on three different frequencies: 900 MHz, 3.4 GHz (via a 2.4/3.4 up/down converter), and 802.11g (2.4 GHz). We feel that these three frequencies will provide us with redundancy and reliability in our communication link. Each computer has three IP's, creating a total of 6 IP's on the network (excluding IP's of the wireless bridges). Therefore proper routing among the separate links is taken care of by the IP layer and the hardware. One small hardware issue we found when dealing with multiple links is that we could not use the bridges in parallel on a network switch. The bridges are designed to replicate traffic on either side of the wireless connection to provide the illusion of a single LAN. On the ARP layer, bridges pick up packets that are not meant for them. This happens back and forth, creating an avalanche effect. We solved this issue by

installing multi-port Ethernet cards on both ends of our link. Each wireless link corresponds to a different Ethernet interface on the control station side and the robot side.

Our software solution is a user-level transport layer called, "RCom", short for "Reliable Communication". It sits on top of UDP/IP.

The API RCom provides the user is very simple. There is one function to send data over the link, and a callback function that gets called when data is sent to you. All knowledge of the multiple links is transparent to the user. The user need only specify a specific ID number that identifies itself, and an ID number to which to send data on the other end.

RCom is symmetric and must be run on both ends of the link. When an RCom API is initialized, it can be initialized as inter-thread or client. In the inter-thread model, RCom is automatically initialized. In the client model, another process is the RCom server and the server/client pair talk through a local socket. The API in both cases are identical. Both sides of RCom keep separate estimations of link statistics so that data can be best routed.

2. Implementation

As previously stated, there are multiple links in RCom. The health of these links must be constantly monitored for effective routing. The period of monitoring in RCom is five seconds. If no data is received on the link during this period, the link is labeled as 'down'. To ensure that links do not go down when there is no user activity on the link, heartbeat messages are used. RCom has a heartbeat thread that sends 5 heartbeat requests over one second, then sleeps for another 4 seconds, making a cycle of 5 seconds. When RCom detects a 'downed' link, no data is sent over it until it comes back up as determined by the reception of heartbeat messages.

RCom does not 'stripe' a user's data over the different links. Combining this with the fact that each link has only one path, the system can conclude that for a given data segment, when an out of order packet is received, there must have been packet loss. This greatly simplifies the reception of data.

In order to select which link to send data on, the RCom layer calculates moving averages of attempted throughput and confirmed throughput. Attempted throughput is simply how much data has been sent on a link, while confirmed throughput is how much data has been ACK'ed by the receiver. RCom selects the link with the highest ratio of confirmed to attempted throughput. This dynamically achieves load balancing. If a link gets congested (by other wireless devices) or is overused, its delivery ratio will drop, causing RCom to favor it less than the other links.

When sending data segments that are larger than the maximum transmission unit (MTU), RCom handles data fragmentation itself, and does not rely on IP fragmentation. Our experimentation showed that this type of fragmentation allows no temporal spacing of the individual packets, reducing the probability of delivering an entire segment almost to zero. When a link sends packets too fast, it essentially clogs itself, creating a need for customized congestion control.

2. Customized Wireless Congestion Control

One of the main goals of this project was to test different congestion control algorithms. We implemented and tested three such algorithms: Fixed Speed, Additive Increase and Decrease (AID), and Additive Increase / Multiplicative Decrease (AIMD). We controlled the speed of a connection by introducing a small latency between individual UDP packets. The duration of this latency inversely determines the (attempted) speed of the link.

2.1 Fixed Speed

A simple way to set the inter-packet time is to use the theoretical transmission limit for each link. The theoretical limits for our three links were calculated based on a packet size of 1500 bytes, and are summarized in Table 1. The number of image frames per second that we can achieve is also calculated, based on a 32869 byte, 640 x 480 jpeg compressed image (at 25% compression).

Link	Freq.	Mbps	ms/packet	FPS
1	3.4 GHz	11	1.6	42
2	2.4 GHz	54	0.216	205
3	900 MHz	1.5	7.79	5.7

Table 1 – Theoretical Optimal Speeds.

2.2 AID (Additive Increase and Decrease)

Part of the problem with using TCP on wireless is due to its "Additive Increase – Multiplicative Decrease" nature. When a 3-DUP is detected (inferred as a packet loss), the speed is reduced by half, mainly to achieve fairness among the users of the network. Since a wireless connection has a lot of random packet losses, this causes constant minimization of speed; by exponentially slowing down, it is almost guaranteed to stay away from its full potential. On the other hand, the increase of speed is a constant amount, not enough to compensate for the random wireless packet losses.

Our approach is to simply use additive instead of multiplicative decrease. In our application, fair share of bandwidth is irrelevant.

2.3 AIMD (Additive Increase / Multiplicative Decrease)

This has the exact same implementation as AID, but instead of decreasing the speed by a constant (add a constant to the inter-packet time), we divide it by a constant (multiply the inter-packet time by a constant). This implementation is intended to mimic TCP.

3. Other Algorithmic Improvements

The code base for RoboCup rescue team had not been modified since May 2006. RCom code was wide open to improvements and algorithmic tweaks.

3.1 Send Queue

A mistake overseen by the initial programmers was the lack of true parallelism between the links. An RCom send was blocking which meant applications that sent data had to wait during the inter-packet sleep. With multiple applications using RCom this is not a problem. But with a single application, the system was unable to transmit data on multiple links concurrently. It selected different links for different segments, only creating the illusion of concurrency.

We addressed this problem by creating a sending queue for each link. Each queue has a dedicated consumer thread that takes packets from the queue, sends it, and sleeps the inter-packet time. Applications simply place a packet on the queue and do not have to wait for the sleep. They only block if the queue is full. If an application is slowing down because of full queues, we do not think this is a problem because the network capacity is simply the bottleneck and speeding up the application further would not make a difference.

3.2 Probabilistic Selection of Links

RCom commits to a link for a given data segment and sends all fragmented packets on the same link. It selects the link with the highest ratio of confirmed to attempted throughput. The problem is that a link that is doing slightly better than the others will constantly get picked, preventing balancing the network load properly. In the worst case, it can cause bursty performance, where one link suddenly gets a lot of data to send, fails at delivering all the attempts. Then the system would switch to another link, repeating the same problem.

We fixed this potential problem by using a probabilistic approach. Each link is probabilistically selected based on its ratio. For example, if one link has a ratio of 0.4 and the other two links have 0.3, the first one will get picked 40% of the time and the others 30% of the time. This should result in a more balanced usage of the network.

3.3 Retransmission of Dropped Packets

RCom never retransmitted dropped packets in order to complete a data segment. Upon receiving a notification of a missing fragment, it simply discarded the rest of that data segment and started transmitting the next one. For drops that happen close to the end of a data segment, this is a serious waste of effort. We implemented

retransmission of such dropped packets whenever the sender has not yet moved on to the next data segment (at the time it receives the notification).

3.4 Other Improvements

There were bugs regarding throughput calculation which we detected while collecting data. We also corrected threading-related bugs, such as incorrect use of mutexes.

4. Experimental Results

4.1 Test Environment

On the sending side (the robot), we have a P4 Mobile 2.0GHz running on a mini-itx motherboard with 1.0 GB RAM. A 4-GB compact flash card is used as a hard-drive with a IDE-CF adapter. The receiving side (operator control station) has a Pentium 4 – 2 GHz CPU, 1GB RAM and a 14GB SCSI hard drive. Both computers have a 4-port PCI Ethernet card.

4.2 Experiment Method

We tested RCom in a similar manner to its intended use in RoboCup, where there is mostly one way traffic of image frames. We wrote a simple program that uses the RCom library to continuously send a 32869-byte jpeg at 50 Hz. We run the system until the receiver successfully receives 1000 images. For the fixed speed method, we used the inter-packet times calculated in Table 1.

We tested the performance of AID, AIMD, and the fixed theoretical approach. We also ran a fourth case: where we looked at the plots of the AID run, and selected inter-packet times that produced the best throughput. We tried the fixed approach with these inter-packet times. We call this method, "Fixed – Observed Optimal".

4.3 Results

The FPS measurements for the four different approaches are summarized in Table 2. AID did considerably better than any of the other approaches. It was able to achieve 10 FPS over a span of 1000 frames. Changing the optimal time did not make much of a difference; in both cases the FPS was a little more than 3 FPS.

Method	**FPS**
Fixed - Theoretical Optimal	3.3
AID	10.1
AIMD	0.2
Fixed - Observed Optimal	3.2

Table 2 – Throughput performance of different approaches.

4.3.1 Fixed-Theoretical Optimal

We considered this the baseline test. Here is a plot of the ACK'ed throughput on the robot side. The total throughput indicates that we could have achieved a much higher frame rate.

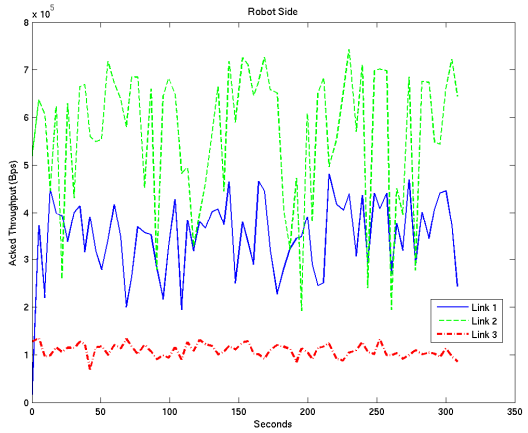

Figure 1 - Fixed-Theoretical Optimal Throughput

4.3.2 AID

Here are plots of the inter-packet times and ACK'ed throughput for our AID test. It illustrates how the adaptive packet spacing changes the throughput.

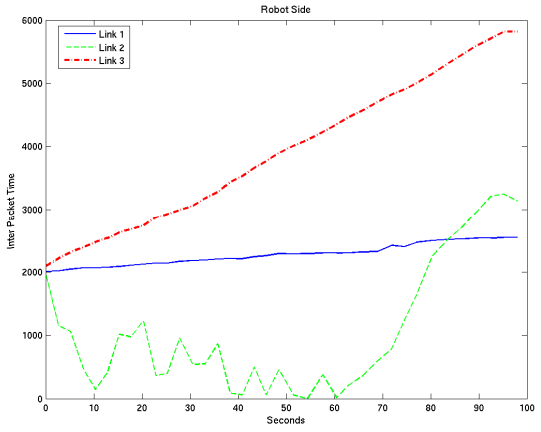

Figure 2 – AID Inter-Packet Times

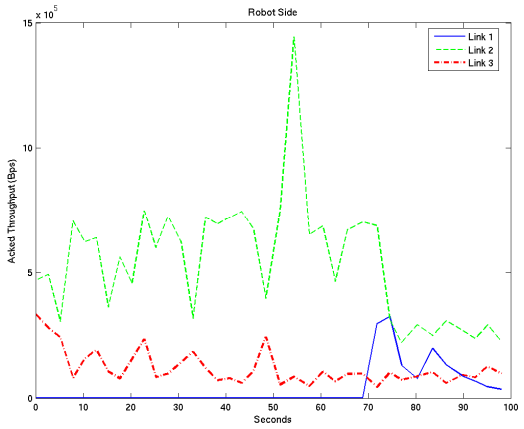

Figure 3 – AID Throughput

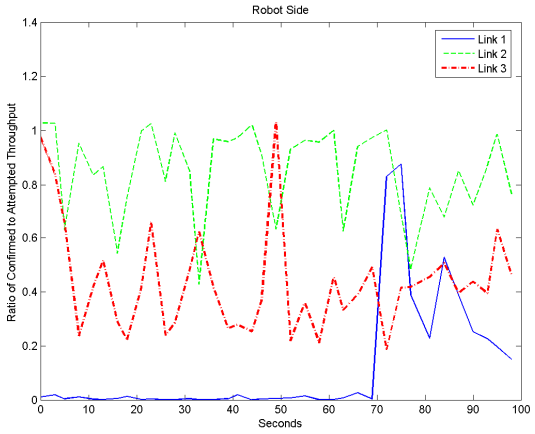

Figure 4 – AID Confirmed to Attempted Ratio

One thing to note from these plots is the drop in throughput near the end of the test. We think this is due to the under-utilized Link 1 suddenly having some ACK'ed packets. This caused the confirmed to attempted packet ratio to jump sharply. This, in turn, caused RCom to favor this link over the superior Link 2. We will continue to investigate this phenomenon. One simple fix would be to smooth the ratio, or put a cap on how small the number of confirmed packets can get so we don't get huge 'divide by 0' ratios.

4.3.3 AIMD

AIMD performed poorly as expected. Reducing the speed by half (as it is done in TCP) slowed down the sender so quickly that the process was hanging for long periods due to long sleeps and the receiver simply stopped receiving data. We then tried reducing the speed to 83% instead of 50% (multiplying inter-packet time by 1.2). Even then, performance was very poor compared to the other approaches. We suspect that the multiplicative decrease made the inter-packet times so large that hardly any data was getting through. The inter-packet times this implementation settled on were about 2 orders of magnitude larger than our other implementations.

4.3.4 Fixed-Observed Optimal

Because of the high performance of AID, and poor performance of the theoretical optimum static case, we sought to determine if our calculated inter-packet times were just wrong. We looked at the plot of AID's total throughput and found when it had peak performance.

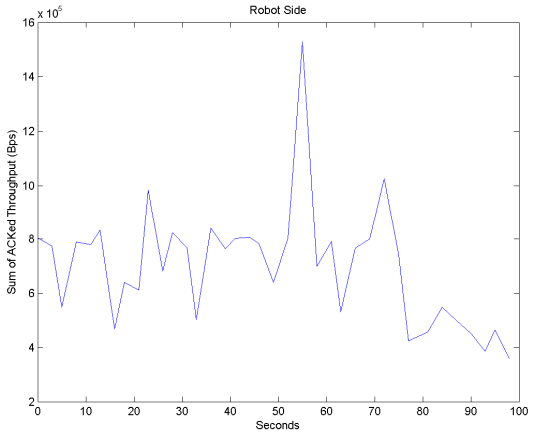

Figure 5 – AID Total Throughput

We then determined the observed optimal inter-packet times (from Figure 2) to be:

Link	Freq.	ms/packet
1	3.4 GHz	2.3
2	2.4 GHz	0.15
3	900 MHz	4.01

Table 3 – Observed Optimal Inter-Packet Times

To our amazement, this test actually performed slightly worse than the calculated theoretical case, achieving only 3.2 FPS. It seems that an adaptive congestion control strategy is essential to high throughput over wireless.

4.3.5 Other Observations

It should be noted that more efficient congestion control algorithms will transmit less

data, faster. This can be seen in Figure 6. Here the sum of received packets is plotted for both AID and the calculated optimum. We see that AID received the 1000 frames in less time, and with fewer packets.

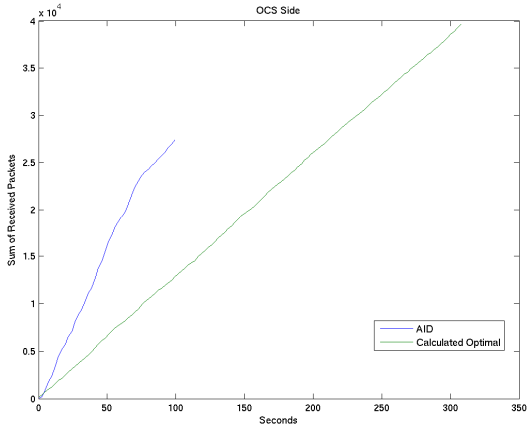

Figure 6 – Received Packets, AID vs. Calculated Optimal

AID is quick to notice downed links, and adjust itself. In this test we only had 2 working links, and part way through the test we intentionally downed a link by removing the Ethernet cable from the wireless bridge. You can see that the attempted throughput drops very quickly, and the overall throughput only takes a small decline.

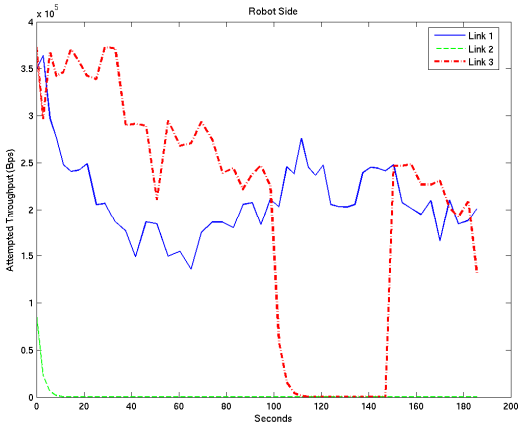

Figure 7 – Link Drop Test – Attempted Throughput, Robot Side

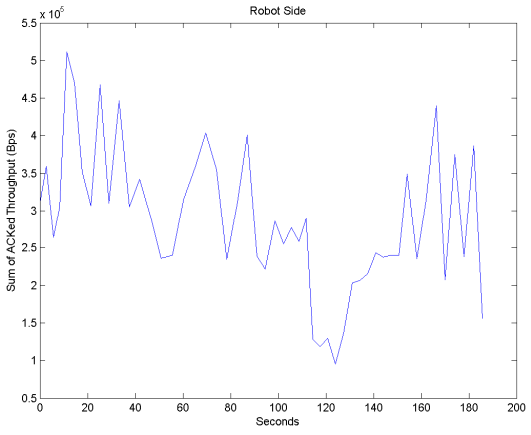

Figure 8 – Link Drop Test – Total Throughput, Robot Side

For certain links, the round trip time on the receiver was much higher than the sender. This is due to queue contention on the sender between ACKs and data segments. The receiver has no data waiting on the queue, so ACKs are sent almost immediately after they arrive. However, on the sender, the ACK's may have to wait on a full queue, only to get placed on the tail of the queue. This is best illustrated by Link 3 during the calculated optimum test as seen in Figure 9. The other links in this test did not have this problem because their queues were not as full.

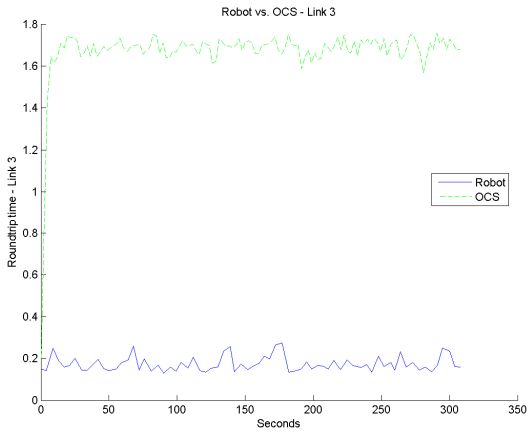

Figure 9 – Link 3 RTT, Calculated Optimal Test

Slightly related to the situation above, another interesting observation is the relationship between inter-packet times on the sending end, and RTT times for the corresponding link on the receiving end. Link 3 has the highest inter-packet time on the sender, and it also has the highest RTT

times on the receiver. The amount of wait in the queue is a linear function of the inter-packet time.

5. Conclusion

Our AID approach performed better than the baseline of using a fixed speed, no matter how optimal that speed is, showing that it is essential to adapt to the status of the wireless even in arrangements where there is no explicit share of the bandwidth. Furthermore, we believe we demonstrated that using an additive instead of multiplicative decrease in speed was more appropriate for our wireless setup.

We have also made substantial improvements to the quality of the code as we described in Section 3. After our implementation efforts, during the experiments, we never observed the CPU utilization of the process to be more than 0.7%.

Making improvements to this communication work was a non-trivial task. The main implementation class has around 1200 lines of code. With additional files, the RCom package has around 3200 lines of code.

The 3.5 GHz link did not have an antenna, which probably contributed to its poor performance.

6. Future Work

Some possible methods to test include:

- Signaling the user application when there is a full queue, so it can potentially slow down the sending.
- Striping data across links.
- Using the slowest link for bundled ACK messages as suggested in [2].
- Sending redundant identical data on all links (based on the slowest link) to eliminate packet loss.
- Migrating the queue of a link that went down. What has already been placed on a queue when a link is declared down is still sent on that link. A solution is to move this data on to another queue.
- Eliminating out-of-order data reception. Because the wireless bridges are different speeds, the receiving side can get data out of order. This can be fixed by using a sequencing system across all the links, in addition to within a single link.
- Implementing intelligent ways of distinguishing between random packet losses and congestion related losses as done in WTCP [3] and in [4].

References

[1] Ye Tian; Kai Xu; Ansari, N., "TCP in wireless environments: problems and solutions", Communications Magazine, IEEE Volume 43, Issue 3, March 2005 Page(s):S27 - S32

[2] Kyasanur, P. Padhye, J. Bahl, P., "On the efficacy of separating control and data into different frequency bands", Broadband Networks, 2005 2nd International Conference, Oct. 2005

[3] Prasun Sinha, Narayanan Venkitaraman, Raghupathy Sivakumar, and Vaduvur Bharghavan. "WTCP: A reliable transport protocol for wireless wide-area networks". TIMELY Group Research Report, January 1999.

[4] L. Magalhaes and R. Kravets, "Transport Level Mechanisms for Bandwidth Aggregation on Mobile Hosts", Network Protocols, 2001. Ninth International Conference, Nov. 2001