Active Networking for TCP over Wireless

Seong-Kyu Song and Scott M. Nettles

Electrical and Computer Engineering Department
The University of Texas at Austin
{sksong,nettles}@ece.utexas.edu

Abstract. TCP assumes that packet losses are due to congestion. Unfortunately, for the increasingly common case of wireless last hops, this may not be the case. The result is poor TCP performance. There has been significant research into this problem, but the solutions either require widespread changes to the network, or are architecturally limited.

Network evolution of this sort is exactly the target of Active Networking (AN). We claim that if some network nodes were AN capable, the range of feasible and deployable solutions to this problem would be greatly increased. We support our claim by presenting a model and architecture of how AN might be deployed to address this problem. We then use this model and architecture to motivate a series of concrete implementations that address various aspects of the problem. These include an implementation of adaptive link control and of the Snoop protocol.

Keywords: TCP, wireless, Active Networking.

1 Introduction

The Transmission Control Protocol (TCP) is one of the central protocols of the Internet. Further, the widespread availability of IEEE 802.11 wireless LANs have made networks in which at least the last hop is wireless common. TCP's congestion control mechanisms assume that all packet losses are caused by congestion. Unfortunately, for wireless links, this is a poor assumption. The result is that over the many networks with wireless hops TCP performs poorly [1].

An obvious solution is simply to do the necessary research to understand how to mix TCP with wireless links and then deploy those solutions throughout the Internet. Not surprisingly, the first step, researching a solution has had significant progress [2,3,4,5,6]. The most general solutions require updating both the TCP implementations on the end hosts and at least some of the routers handling wireless traffic. Unfortunately, in today's Internet, such an update is very difficult to achieve. As a result, there has also been significant work on solutions that do not require updating the end hosts (or perhaps only the one connected wirelessly), essentially restricting the design space to transparent modifications of the basestation connecting the wired network to the wireless one [4,5]. Unfortunately, this architectural restriction can have adverse performance implications [7].

The goals of Active Networking (AN) are to facilitate network evolution and customization. Our claim is that if we had a network that incorporated AN in

G.J. Minden et al. (Eds.): IWAN 2004, LNCS 3912, pp. 156–168, 2007.

at least some its nodes, the range of feasible and deployable solutions to the problem of TCP over wireless would be greatly increased. Of course, if AN in its most general form penetrated everywhere, it would clearly solve this problem, because it would be easy to simply deploy the best, most general solutions and as new solutions were developed to deploy them. Here we are interested in exploring the implications of more limited AN penetration on possible solutions.

In addition to presenting background material on TCP over wireless and our AN approach and platform, we make our case in two ways. First, to a large extent the issues at hand have to do with implementation architecture. Thus a key part of our argument is a presentation of a model of TCP over wireless systems that lays out the possible design space, followed by a consideration of the high-level architectural issues. Second, to make things concrete, we present implementations of some of the possible solutions.

To implement our demonstrations, we have chosen to use our Mobile Active Network Environment (MANE) [8]. MANE is the most recent embodiment of our work on PLAN [9,10] and is a direct decedent of PLANet [11]. Like our earlier work, MANE combines programmable Active Packets with downloadable node resident Active Extensions. MANE goes further than any of our PLANet implementations in its support for Active Extensions since it provides not just for "plug-in" extensions, but also for "dynamic-update" extensions [12,8].

The remainder of this paper is organized as follows. Section 2 presents background material on TCP over wireless. Section 3 presents our model and architectures and fundamentally addresses the question "How can AN Help?" Sections 4 and 5 present specific implementations and discusses alternative approaches. Section 6 presents performance evaluation and Section 7 concludes the paper.

2 TCP Background

TCP is a connection-oriented transport layer protocol responsible for end-to-end reliable data transmission. There are several problems with TCP's functionality and performance over wireless links. Over such links, there may be more fluctuation of bandwidth and delay than in typical wired networks, stressing TCP's ability to adapt. A key problem of TCP over wireless links arises because TCP's error recovery and congestion control are closely coupled due to the assumption that packet drops are only caused by congestion. Although this assumption is valid over wired links, wireless links are lossy and cannot be assumed to be reliable despite their link-level error recovery schemes [13]. These points are reinforced in the literature, where it has been shown that TCP's performance significantly degrades over wireless links [1]. A variety of solutions have been proposed for the problem of TCP over wireless links [2]. They can be classified into two main categories: *End-to-end* and *Transparent*.

End-to-end solutions require modifying TCP at both end points while maintaining end-to-end semantics [2,3,14,15,16]. These approaches are based on the distinction between congestion losses and corruption losses [14,17]. The drawback of these approaches is that they require fundamental changes to TCP on the

hosts. The need to replace already deployed versions of TCP means that deployment of these approaches will be difficult and slow. Besides, this approach needs more care because it is unclear that the modified TCP will perform well both on wired and wireless links. From experience, we see that it may take time to find problems of newly deployed protocols that were thought to be well-designed.

In the transparent approach, link-level losses are hidden from the transport layer [4,5,6]. These approaches attempt to improve TCP performance either by using enhanced link control schemes [4] or by utilizing Performance Enhancing Proxies (PEP) [18,5,6]. Therefore, existing hosts operate normally without knowing whether the connection is over wireless or wired links. The main advantage of these approaches is that they can more practically be deployed incrementally. It is easier to modify link-layer protocols on the nodes with wireless links than the TCP protocol deployed on every end-host. However, as we will see from the TCP snoop protocol, link layers may be aware of the transport layers' semantics and session state information. In addition, this approach has the possibility of redundancy, inefficiency, or even ineffectiveness [7].

3 How Can Active Networking Help?

Our goal is not to devise fundamentally new schemes for solving the basic problem of TCP over wireless links, but rather to show how AN could be used to help to implement and deploy existing schemes. At a high-level, this is an architectural question, where and in what form can AN be useful? To answer this question, we begin by creating a model of the underlaying system. The section concludes by considering a variety of architectures that map AN capabilities on to this model. The rest of the paper is principally an exploration of some specific instances of these mappings.

3.1 Model

The model of a TCP session shown in Figure 1 captures many of the key architectural issues. Communication is between a Mobile Host (MH) and a Fixed Host (FH). A Base Station (BS) connects the wired network where the FH resides to the wireless one where the MH resides. Unlike most of the related work discussed above, we include the case where the MH may need multiple wireless hops to reach the BS. Also, the related work focuses on the case where the bulk of the data is being transmitted from the FH to the MH. In general, we are also concerned with the case where the MH is the primary source of data.

Figure 1 also illustrates some of our thinking about where and what kind of AN technology might be deployed. We subscribe to the SwitchWare [9] architecture of AN, in which there are both active packets (APs) containing executable code and active extensions (AEs) which are downloaded dynamically to modify or extend nodes. We assume that we have full control of the MH and thus can expect that both APs and AEs can be used there when needed. Similarly, we entertain the possibility that the wireless network is "all active" and thus that

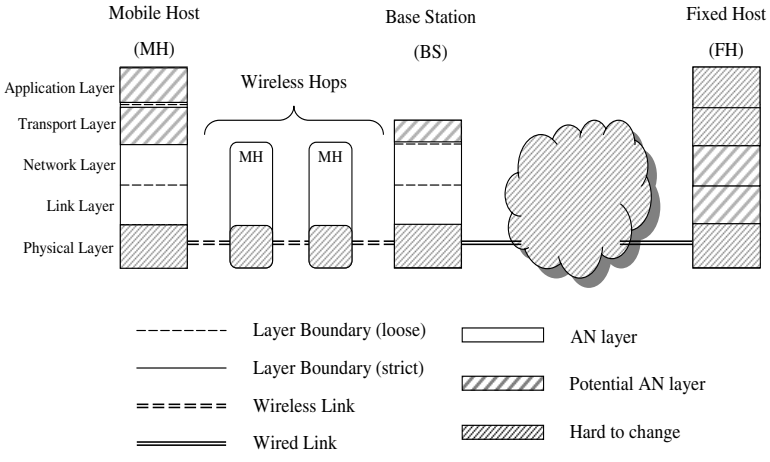

Fig. 1. AN model for TCP over wireless

we could potentially deploy both AEs and APs there. Three possibilities exist for the BS. First, if the BS can employ no activeness, then we are restricted to end-to-end solutions (and must have an active enabled FH). Second, perhaps for security reasons, the BS may allow AP processing, but not allow AEs to be downloaded. Finally, the BS may support both APs and AEs. The intermediate links between the BS and FH are not a source of the problems we are trying to address and so without loss of generality, we can assume they are not "active." However, in the case that the BS is not "active," some BS-centric approaches will work if deployed at an intermediate node. Finally, the FH has the same basic options as the BS. Of course, it is likely that a MH will have more control over the BS than the FH, so it is likely that in practice the FH will allow fewer "active" options than the BS.

Finally, Figure 1 also touches on the issue of layer-crossing. The problems we are addressing come fundamentally because TCP violates the basic layering principles of the network by making an incorrect assumption about the nature of the wireless physical layer. Thus it is not surprising that many of the approaches to solving these problems also violate layering. In fact, one of our premises is that since AN can support flexible controlled layer-crossing, it is well suited to these solutions. Thus the figure shows which layers we expect to be the most "permeable" as well as at which layers we most expect to deploy either APs or AEs. One case that is not illustrated is the use of "shim" layers. These are simply layers that are inserted between existing layers.

3.2 Requirements, Architecture, and Capabilities

Given the basic system model, we state two system requirements, consider the possible architecture of solutions and discuss several AN capabilities that potentially play an important role in the solution space.

The first requirement is preservation of TCP's end-to-end semantics: reliable, in-order, duplicate free delivery. We view this as a strict requirement of any solution; taking the view that these semantics define what TCP is and that any system that does not provide these features is not TCP. The second requirement is backward compatibility. Since in some scenarios the possibility exists of using AN to modify the end hosts implementation of TCP, we do not view this as a strict requirement. However, many other scenarios exist that deny this possibility and so it is important to consider. Since we view the MH as fundamentally more changeable than the FH, backward compatibility issues focus at the FH. Backward compatibility then takes two forms. First are systems where the FH is "active," but the TCP implementation is not. Such systems admit end-to-end approaches, but must mask any "activeness" from the TCP layer. Second are systems in which the FH is unchangeable and transmits standard TCP segments. In this case, any "activeness" must be masked before the FH. Given our assumption that the Internet is not active, this means "activeness" must be masked at or before the BS. In general, we would like to be able to support "islands" of AN functionality isolated by conventional networks. We will illustrate how this may be done in Subsection 4.3.

In our view, there are two basic architectural approaches: horizontal and vertical. The horizontal approach works between peer layers and does not cross layer boundaries. For example, link layer protocols over wireless hops can adaptively cope with fluctuating channel conditions and reduce link-level errors. An important special case is when the peer layers are dynamically inserted (and removed) shim layers. This is essentially the idea of Protocol Boosters [19]. In Section 4, we will discuss how our implementation system makes this idea especially useful. In contrast, the vertical approach allows layering violations and information sharing between layers. For example, the BS is allowed to cross layers in dealing with TCP-aware processing. To avoid congestion control on end hosts, the BS attempts to foil fast retransmit by adaptively manipulating duplicate ACKs.

One of the key AN capabilities that can be leveraged to assist us is the ability to adapt quickly, perhaps even on a packet-by-packet basis. This ability derives from the fact that the code (or data used by that code) contained in APs can change in each packet. We will show an example of this based on link error control in Subsection 4.3. A final AN capability of importance centers on AEs. AEs can be dynamically downloaded and can add to or modify the behavior of node resident code. The implication of this is that even protocols that need new or modified node resident functionality can be incrementally deployed on-the-fly. As an example, consider a MH that wishes to use an enhanced protocol that requires node-resident functionality at the BS. Assuming the BS supports AEs, then the MH can simply extend the BS. Essentially BS has been adapted to support the new protocol.

4 Horizontal Adaptive Link Error Control

One obvious approach is simply to improve the error characteristics of the wireless link. As our first example, we consider how to use the horizontal approach to

```
1: fun checkCRC(chk, crc) =
2:  let val crcCalcul = crc32(chk)
3:      val nexthop = defaultRoute(getSrc()) in
4:  ( if(crcCalcul = crc) then (
5:      eval(chk);
6:      OnNeighbor(|deQueue|(), #1 nexthop, getRB(), #2 nexthop)
7:    ) else ()
8:  )  end
9:
10: fun arq(dst, chk) =
11:  let val crc = crc32(chk)
12:      val nexthop = defaultRoute(dst) in
13:  ( enQueue(|checkCRC|(chk, crc), #1 nexthop);
14:    OnNeighbor(|checkCRC|(chk, crc), #1 nexthop, getRB(), #2 nexthop)
15:  ) end
```

Fig. 2. PLAN for basic ARQ

implement this idea. The tricky issue is that how best to do this is a function of the link error rate, which is changing dynamically. If the error rates are very low, it might make sense to have no link-level error correction. At higher, but still moderate error rates, a basic ARQ scheme is employed because of its simplicity and low overhead. However, as error rates increase, frequent retransmissions degrade performance. Thus at high rates, to control errors more efficiently, FEC is added into ARQ. By combining two coding procedures, hybrid ARQ/FEC can get the benefits of both [20,21]. In this section, we show how PLAN/MANE can be used to implement this adaptivity using a shim layer. For adaptive link error control, we place the shim layer between the link layer and network layer.

4.1 Basic ARQ

We begin with a simple ARQ scheme. For simplicity, we assume we have only one wireless hop. Thus we expect the round-trip times seen by the link-level ARQ to be small. Therefore, we adopt an idle RQ or *stop-and-wait ARQ* scheme rather than a *selective-repeat ARQ* or *go-back-N ARQ* scheme [22]; however it would be straightforward to include other ARQ schemes when desirable. In that case, we would not need to change the node-resident services, but would use a different PLAN program containing the required ARQ algorithm.

Figure 2 shows the PLAN code for our ARQ scheme. This code contains the ARQ scheme in chunks, such as CRC calculation, timeout and retransmission, and ACK reply. For error detection, the sender calculates a CRC-32 (Line 11) and sends a new chunk (`checkCRC`) containing the original chunk and the corresponding CRC (Line 14). It also stores the packet in the interface queue for retransmission (Line 13). The destination evaluates the chunk, thus evoking `checkCRC`, which executes to compute the CRC of the received chunk and comparing it with the original CRC (Lines 2,4). If the results are the same, the original chunk is evaluated on the destination (Line 5). The destination is also

required to generate a chunk to invoke the `deQueue()` function on the sender. This chunk works like an acknowledgment and frees the packet in the interface queue (Line 6). Note in practice, this ACK chunk might also implement other functionality as well, such as updating an RTT estimate. This particular code is specialized for a single wireless hop because it always does the CRC check on its neighbor. However, it could be used from either the BS or the MH. Further, it would be easy to generalize this approach to support multiple wireless hops. In this case, if transmitted from the BS, it would simply defer execution of `checkCRC` until it reached its final destination and it would also need to carry with it the address of the BS to provide the "ack" with a destination.

4.2 ARQ/FEC

By utilizing ARQ/FEC at high error rates, we can maintain constant throughput at the expense of encoding/decoding overhead and complexity. The code for ARQ/FEC is similar to that for the basic ARQ. The key difference is that before the original chunk is transmitted it is encoded using Reed-Solomon coding and then when it is received, it is decoded. The code for this case can be found in Song [23]. By including the FEC strength in the chunks, we could control the level of error correction on a packet-by-packet basis.

4.3 Adaptive Link Control

Given basic ARQ and ARQ/FEC the question is how to combine them so that the appropriate one is used based on the quality of the channel. Figure 3 presents code which does this when sent from a FH. It depends on the BS to identify itself by returning true when `isThisHostBS` as well as to maintain a measure of channel quality, queried by `isChanGood`. The basic idea is that the packet single hops through the network (Lines 2,17) looking for the BS (Line 3). At the BS, it queries the channel state (Line 4) and if it is good, it uses no error control scheme (Line 5). If the channel is not as good, it uses either basic ARQ (Lines 9,10) or ARQ/FEC (Lines 13,14). Note `checkCRC` is the same as the previous code and `decode` is used for RS decoding. The fact that the algorithm is encoded in the packet means that we can apply this adaptation on a packet by packet basis. This is quite similar to protocol boosters, except that the packet itself decides whether "boosting" is needed or not.

4.4 AN for Channel Monitoring

For adaptive link control, we need to track the state of the channel. One approach is for the sender to use ACKs (or rather their lack) to tell when the channel is bad. With AN it is easy to do better. The key observation is that the receiver is in the best position to monitor the channel, while the sender is the one that needs this information. Assuming the receiver records channel information, we can use APs to query this state.

Figure 4 shows the code for an out-of-band channel monitor. The function `getChanInfo()` (Line 2) defines a standard interface to get information on channel characteristics. This function returns various channel information depending

```
1: fun adapLink(dst, chk) =
2:  let val nexthop = defaultRoute(dst) in
3:  (if(isThisHostBS()) then (
4:     if(isChanGood(#1 nexthop)) then
5:       OnNeighbor(|noControl|(chk), #1 nexthop, getRB(), #2 nexthop)
6:     else (
7:       enQueue(|adapLink|(dst, chk), #1 nexthop);
8:       if(isChanSoSo(#1 nexthop)) then (
9:         let val crc = crc32(chk) in
10:          OnNeighbor(|checkCRC|(chk, crc), #1 nexthop, getRB(),
11:                     #2 nexthop)  end )
12:      else ( (* if channel is worse, use RS coding *)
13:        let val codeword = fecEncoding(chk, "RS", 255, 223) in
14:         OnNeighbor(|decode|(codeword, "RS", 255, 223), #1 nexthop,
15:                    getRB(), #2 nexthop)  end ) ) )
16:  else
17:   OnNeighbor(|adapLink|(dst, chk), #1 nexthop, getRB(), #2 nexthop)
18:  ) end
```

Fig. 3. PLAN for Adaptive Link Control

```
1: fun report(indicator) =
2:   let val measure = getChanInfo(indicator)
3:       val src = defaultRoute(getSrc()) in
4:         OnNeighbor(|print|(measure), #1 src, getRB(), #2 src)
5:   end
6:
7: fun probe(dst) =
8:   let val nexthop = defaultRoute(dst) in
9:     OnNeighbor(|report|("RSS"), #1 nexthop, getRB(), #2 nexthop)
10:  end
```

Fig. 4. PLAN for Monitoring RSS

on the parameter, `indicator`, such as the Received Signal Strength or Signal-to-Noise Ratio. The code composes a query and sends it (Line 9). The query executes on the receiver and returns the result to the sender (Line 4). Note that this is much more flexible than the conventional approach, which would require specifying a special packet format and protocol for such queries.

An important variation would be to piggyback the query chunk on a data packet. This is easy to do because chunks are data and it is easy to compose various chunk oriented calculations. The result is that such queries can be done without sending additional packets and yet remain transparent to the data flow. This ability to piggyback control on data transparently, solves a key problem with Protocol Boosters [19], controlling when to add or remove a booster.

Finally, consider a system like IEEE 802.11 which precedes each data transmission with a request-to-send (RTS)/clear-to-send exchange(CTS). Then the

RTS could act as a channel probe, while the CTS could return the channel state to the sender, which would then be able to choose a error correction scheme or FEC strength. In a conventional network, this would require changing the format and function of the RTS and CTS. In however, if the wireless link sent APs for its RTS and CTS, then adding to or modifying the function of these parts of the protocol would become just a matter of packet programming.

5 Vertical Snoop Protocol

Even with adaptive link error control, packet drops may be still possible and the resulting congestion control action can cause performance degradation. In vertical adaptivity, collaboration and information sharing across layers on the BS are allowed to adaptively control the TCP flow. We claim that AN is advantageous because AN facilitates cross-layering implementation by allowing layer-specific information to be included in active packets.

In this approach, the lower layer protocols on the BS are aware of TCP semantics and adjust TCP flow information to prevent congestion control from taking place due to packet drops over wireless links. Further, by following up the parts of the end hosts' TCP Control Block (TCB) [24], the BS can take actions on incorrect congestion control, such as adjusting RTT measures and screening three duplicate ACKs. Using packet programming, we can deploy the snoop protocol onto the BS, which can improve performance of TCP connections from fixed hosts (FH) to mobile hosts (MH).

We have implemented these ideas in the form of a PLAN/MANE implementation of the snoop protocol, but space does not permit us to exhibit the code. It can be found in Song [23]. Substantial parts of the snoop protocol are implemented in active packets and this example shows how to easily deploy a new protocol. There is no need to update protocol stacks on the BS. Service extensions on the BS mainly implement the cross-layering mechanisms. As an adaptation layer, the service extensions transform active packets to TCP segments or vice versa. Evaluation of the PLAN packet on the BS actualizes the snoop protocol and enhances TCP performance over wireless links. This is fundamentally different from the Proxy Transport Service [25] in that our approach is transparent and maintains TCP semantics.

6 Performance Evaluation

In this section, present our performance evaluation, starting with some details of our implementation and the network setup. We then present a comparison of our adaptive link protocol with the nonadaptive protocols it is composed of. We conclude with an evaluation of our snoop protocol.

6.1 Evaluation Setup

To support our current experiments required some additions to the version of MANE discussed in [8]. The most significant was an implementation of an active

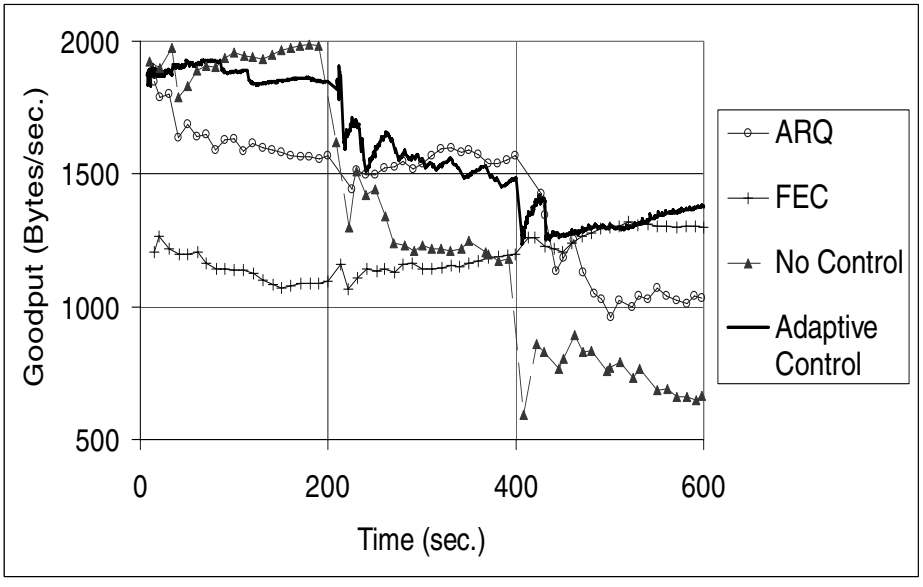

Fig. 5. Comparison of Link Error Control Techniques

version of TCP. To achieve this, we added a data structure called the Transmission Control Block (TCB) [24]. Each TCP host maintains information about TCP connections and TCB is used to store this information. Among the variables maintained in the TCB, we added the basic ones needed for the congestion control and the sliding window protocol, which included sequence numbers, round trip time (RTT) measures and variance, timeout values for retransmission, and the congestion window. In addition to TCP, we added the functions for calculating a 32-bit Cyclic Redundancy Check (CRC-32) to our frames as a Frame Check Sequence (FCS) and for Reed-Solomon encoding/decoding [26].

In all of our experiments, we used the same network topology. The MH and the FH are connected through the BS. The MH is one wireless hop away from the BS, and the BS is connected to the FH through one router using wired links. We emulated the lossy wireless channel by randomly changing bits in packets. The number of corrupted bits is determined by the channel's Bit Error Rate (BER); the channel's BER is changing on a scenario basis.

6.2 Active Link Error Control

In Figure 5, we present a performance comparison of four link error correction schemes: no error correction, ARQ, ARQ/FEC, and the adaptive hybrid protocol shown in Figure 3. Because we wanted to demonstrate adaptivity, rather than experiment with channel monitoring, we had our channel monitoring functions return the correct value, rather than trying to estimate it dynamically. In this case, the MH is the TCP sender.

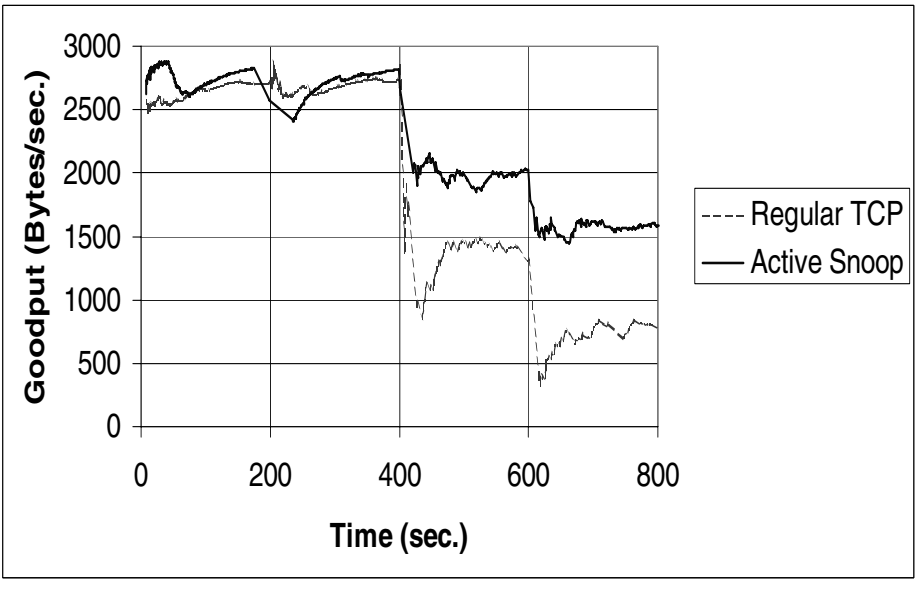

Fig. 6. Comparison of Regular TCP and Active snoop protocol

Figure 5 has time along the X-axis and goodput along the Y-axis. Initially, the channel is not lossy, but its error rate increases at time 200 and then worsens further at time 400. The individual protocols perform as we would expect, with the no error correction case showing the strongest dependence on the error rate and ARQ/FEC showing the least. The more interesting result is for the adaptive protocol. At low error rate it equals or sometimes betters the performance of ARQ. When it does better it is actually doing no error correction. When the error rate increases, it is able to detect this and adapt using ARQ or ARQ/FEC, respectively.

6.3 Active Snoop Protocol

Figure 6 shows a performance comparison of "regular" TCP when no snooping is done at the BS and with our snoop protocol. In this case, the FH is the sender. Again, time is along the X-axis and goodput is along the Y-axis. Initially, the channel is not lossy, but its error rate increases at time 200 and worsens further at time 400 and 600. As expected, when there are no or low errors, the performance of the two versions is similar. However, as error rates increase the snoop protocol outperforms the regular TCP by suppressing unnecessary congestion control.

7 Conclusion

TCP is not well suited to networks with wireless links. Conventional solutions to this problem are limited by the need to be transparent and backward compatible.

AN can ease these limitations by greatly increasing the possible implementation and deployment strategies. We demonstrated this by first modelling the TCP over wireless system and then showing how AN architectures and capabilities can apply to that model. We then showed a number of implementations that used these architectures and capabilities to help wireless TCP performance. We used MANE to evaluate the performance of these implementations, showing that they behave as expected.

We expect that future work will focus on further exploring our two architectural styles. For example, as one of the horizontal approaches, the network layers of the MH and the BSes could adaptively change paths between them so that link fluctuation do not affect the end-to-end flow. We could expand this approach into support for handoff. Another horizontal approach would be to adaptively change the Maximum Transmission Unit of the wireless link, so that smaller packets are sent when the link has a high bit error rate. On the other hand, if the FH is AN-capable, we can develop more efficient adaptive control over TCP flows. We expect to apply this advantage to handling handoff, during which harsh link deterioration and route changes happen at the same time.

References

1. Pentikousis, K.: TCP in Wired-cum-Wireless Environments. IEEE Communications Surveys (2000) 2–14
2. Balakrishnan, H., Padmanabhan, V.N., Seshan, S., Katz, R.H.: A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. In: Proc. of ACM SIGCOMM'96. (1996) 256–269
3. Ramakrishnan, K., Floyd, S., Black, D.: The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168, IETF (2001) http://www.ietf.org/rfc/rfc3168.txt.
4. Ayanoglu, E., Paul, S., thomas F. LaPorta, Sabnani, K.K., Gitlin, R.D.: AIRMAIL: A Link-Layer Protocol for Wireless Networks. ACM Wireless Networks **1** (1995) 47–60
5. Balakrishnan, H., Seshan, S., Amir, E., Katz, R.H.: Improving TCI/IP Performance over Wireless Networks. In: Proc. of ACM MobiCom'95. (1995) 2–11
6. Bakre, A., Badrinath, B.: I-TCP: Indirect TCP for Mobile Hosts. In: Proc. of the 15th Int. Conf. on Distributed Systems. (1995) 136–143
7. DeSimone, A., Chuah, M.C., Yue, O.C.: Throughput Performance of Transport-Layer Protocols over Wireless LANs. In: Proc. of IEEE GLOBECOM'93. Volume 1. (1993) 542–549
8. Song, S.K., Shannon, S., Hicks, M., Nettles, S.: Evolution in Action: Using Active Networking to Evolve Network Support for Mobility. In: Proc. of IWAN 2002. (2002) 146–161
9. Alexander, D., Arbaugh, W., Hicks, M., Kakkar, P., Keromytis, A., Moore, J., Gunter, C., Nettles, S., Smith, J.: The SwitchWare Active Network Architecture. IEEE Network Magazine **12** (1998) 29–36
10. Hicks, M., Kakkar, P., Moore, J., Gunter, C., Nettles, S.: PLAN: A Packet Language for Active Networks. In: Proc. of ACM SIGPLAN International Conference on Functional Programming Languages, ACM (1998) 86–93
11. Hicks, M., Moore, J., Alexander, D., Gunter, C., Nettles, S.: PLANet: An Active Internetwork. In: Proc. of IEEE INFOCOM'99, IEEE (1999) 1124–1133

12. Hicks, M., Moore, J., Nettles, S.: Dynamic software updating. In: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, ACM (2001) 13–23
13. Chockalingam, A., Zorzi, M., Tralli, V.: Wireless TCP Performance with Link Layer FEC/ARQ. In: ICC'99 Proceedings. Volume 2., IEEE (1999) 1212–1216
14. Krishnan, R., Sterbenz, J.P., Eddy, W.M., Partridge, C., Allman, M.: Explicit transport error notification (ETEN) for error-prone wireless and satellite networks. Computer Networks **46** (2004) 343–362
15. Akyildiz, I.F., Morabito, G., Palazzo, S.: TCP-Peach: A New Congestion Control Scheme for Satellite IP Networks. IEEE/ACM Transactions on Networking **9** (2001) 307–321
16. Casetti, C., Gerla, M., Mascolo, S., Sansadidi, M., Wang, R.: TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks. Wireless Networks Journal **8** (2002) 467–479
17. Biaz, S., Vaidya, N.H.: Distinguishing Congestion Losses from Wireless Transmission Losses: A Negative Result. In: Proceedings of the Seventh International Conference on Computer Communications and Networks (IC3N). (1998)
18. Border, J., Kojo, M., Griner, J., Montenegro, G., Shelby, Z.: Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations. RFC 3135, IETF (2001) http://www.ietf.org/rfc/rfc3135.txt.
19. Feldmeier, D.C., McAuley, A.J., Smith, J.M., Bakin, D.S., Marcus, W.S., Raleigh, T.M.: Protocol Boosters. IEEE Journal on Selected Areas in Communications **16** (1998) 437–444
20. Brayer, K.: Error Control Techniques Using Binary Symbol Burst Codes. IEEE Trans. on Communication Technology **16** (1968) 199–214
21. Burton, H.O., Sullivan, D.D.: Errors and Error Control. Proc. of IEEE **60** (1972) 1293–1310
22. Lin, S., Daniel J. Costello, Jr.: 15. In: Error Control Coding: Fundamentals and Applications. Prentice-Hall, Inc. (1983)
23. Song, S.K.: Applying Active Network Adaptability to Wireless Networks. PhD thesis, The University of Texas at Austin (2004)
24. Postel, J.: Transmission Control Protocol. RFC 793, IETF (1981) http://www.ietf.org/rfc/rfc793.txt.
25. Patil, S., Kumar, M.: TCP Enhancement Using Active Network Based Proxy Transport Service. In: Proc. of IWAN 2003. (2003) 103–114
26. Reed, I.S., Solomon, G.: Polynomial Codes Over Certain Finite Fields. Journal of the Society for Industrial and Applied Mathematics **8** (1960) 300–304