2006

InPCM: a network caching technique for improving the performance of TCP in wireless ad-hoc networks

Andrew Adinegara

W. H. O. Lau

Kwan-Wu Chin
University of Wollongong, kwanwu@uow.edu.au

InPCM: a network caching technique for improving the performance of TCP in wireless ad-hoc networks

Abstract

We propose a novel mechanism called In-Network Packet Caching Mechanism (inPCM) to address TCP's poor performance in IEEE 802.11 based multi-hop wireless networks. In particular, we address TCP's inappropriate response to bursty and location dependent errors. The key concept is the use of intermediate nodes to perform packet recovery on behalf of TCP senders, similar to the well-known Snoop TCP but adapted to work over multi-hop wireless networks. We have conducted ns-2 simulation studies over a variety of network conditions and topologies. Our results confirm InPCM's benefits to TCP in terms of delay and throughput. Moreover, it is immediately deployable without modifications to current protocols.

Disciplines

Physical Sciences and Mathematics

Publication Details

Adinegara, A., Lau, W. & Chin, K. (2006). InPCM: a network caching technique for improving the performance of TCP in wireless ad-hoc networks. IEEE Wireless Telecommunications Symposium 2006 USA: IEEE.

InPCM: A Network Cache Technique for Improving the Performance of TCP in Wireless Ad-Hoc Networks

Andrew Adinegara and W.H.O Lau
Curtin University of Technology
GPO Box U 1987
Perth, Western Australia
{12380189, lauhow}@cs.curtin.edu.au

Kwan-Wu Chin
University of Wollongong
Northfields Ave
Wollongong, NSW, Australia 2522
kwanwu@uow.edu.au

Abstract

We propose a novel mechanism called In-Network Packet Caching Mechanism (InPCM) to address TCP's poor performance in IEEE 802.11 based multi-hop wireless networks. In particular, we address TCP's inappropriate response to bursty and location dependent errors. The key concept is the use of intermediate nodes to perform packet recovery on behalf of TCP senders, similar to the well-known Snoop TCP but adapted to work over multi-hop wireless networks. We have conducted ns-2 *simulation studies over a variety of network conditions and topologies. Our results confirm InPCM's benefits to TCP in terms of delay and throughput. Moreover, it is immediately deployable without modifications to current protocols.*

1 Introduction

Wireless Ad-hoc Networks (WANETs) operate without a fixed infrastructure and can be set up in different environments and terrains quickly. To date, WANETs have applications in areas such as battle-field communications, rescue operations [7], and sensor networks [20]. Moreover, it has gained commercial prominence, as demonstrated by Motorola's Mobile Mesh Network [9], which allows one to deploy low cost, high performance and scalable networks.

The Transport Control Protocol (TCP) [22] is the most dominant protocol in use today and likely to remain so in the future. Unfortunately, TCP's behavior and in particular its performance in WANETs leaves a lot to be desired when compared to its performance in wired networks. The reason is because TCP assumes that the network is congested whenever it detects packet loss; rightly so given that link errors in wired networks are relatively rare. In contrast, packet loss in WANETs can be due to factors such as bad channels [23], medium access control (MAC) contention [6] and link breakages [7]. As a result, TCP experiences timeouts and enters the slow-start phase frequently.

TCP's performance issues over wireless networks have spurred many research efforts [20][16][28][27][25][24][13] [11][17] with the common goal of equalizing TCP's performance in both wired and wireless networks. Early works [2, 8] have addressed the high packet loss rate problem by performing packet recovery at base stations. Recent efforts have focused on modifying TCP's behaviors, for example, adjusting its congestion window size [6][13] and employing an end-to-end feedback mechanism [28][19][5]. Other approaches include modifications to the IEEE 802.11 MAC protocol [3][26], and enhancing ad-hoc routing protocols [16][14][15]. However, there has been little research on performing packet recovery in the presence of high packet error rate, specifically in multi-hop wireless networks.

In an early work, [2] propose Snoop TCP, a caching mechanism that is deployed at base stations. The caching mechanism is able to hide losses from TCP senders by having an agent residing at the base station retransmit lost packets. They show that Snoop TCP is transparent to both sender and receiver, and does not interfere with TCP's operation. In light of its 1-hop performance, we ask the following key question: Can Snoop TCP be extended to work in WANETs? Clearly, the main challenge is the absence of base station which to locate Snoop TCP like caching mechanism since all nodes in a WANET can be hosts and routers.

In this paper, we show that indeed Snoop TCP can be extended to work in WANETs. Our solution, called In-Network Packet Caching Mechanism (InPCM), employs caching at intermediate nodes to mitigate packet losses by performing "local" retransmissions. Other features of InPCM include cache optimization, fine-grained retransmission timeout, and ability to operate in conjunction with other technologies such as Link-RED [11, 10] to reduce contention. In addition, InPCM has the following advantages over the underlying MAC's retransmission mecha-

1-4244-0046-5/06/$20.00 (c)2006 IEEE

nism:

- IEEE 802.11 MAC's retransmission is only for 1-hop, whereas InPCM is capable of working across multiple hops.
- InPCM uses a longer timeout value compared to MAC. Thereby, it can avoid persistent congestion.
- InPCM has access to routing information. Thus, allowing it to send packets on alternative routes and bypass neighbors experiencing severe fade.
- InPCM can incorporate different policies for handling failed packets. For example, a policy that intelligently drops packets to reduce contention; see Section 3.9.

We have conducted *ns-2* simulations over various network topologies (e.g., linear, grid and random), and our results confirm the effectiveness of InPCM in improving TCP's performance in WANETs.

This paper has the following structure. Section 2 discusses the background and related works. We start by giving an overview of InPCM in Section 3 followed by a discussion of various design criteria in Section 3.1. After that we present InPCM's eviction policies and retransmission timeout calculation. Section 4 presents our simulation parameters and the network topologies used to verify InPCM. We present our results in Section 5, and Section 6 concludes this paper.

2 Background

2.1 TCP over WANETs

The error prone nature of wireless networks has spurred much research into improving TCP's poor performance in such networks. Representative works include TCP-F [5], TCP-ELFN [19], ATCP [17], and EPLN [28]. Most of these solutions, however, require modifications to TCP so that it becomes aware of its operating environment; error prone wireless channel and MAC behaviors.

TCP-F [5]'s objective is to distinguish between losses due to congestion and route failures. To do this, TCP-F requires two notification packets, Route Failure Notification (RFN) and Route Reestablishment Notification (RRN). The basic idea is to freeze TCP's timer and any ongoing packet transmissions when the TCP sender receives a RFN. Once a RRN packet is received, the sender restores its TCP's timer.

ATCP [17] puts TCP into different states according to feedback it receives from intermediate hops. When the network is partitioned, TCP is put into a persist state where TCP will not retransmit or transmit any packets during this period. TCP is also shielded from non-congestion related losses so that its congestion control/avoidance mechanisms will not be invoked. However, when true congestion occurs, normal TCP's congestion control/avoidance is used.

EPLN [28] relies on notification messages to indicate link failure and packet loss. For example, when a node detects a link failure, the node will send a notification message, which includes information about lost packets. Upon receiving a notification, the TCP sender records the information, disables its retransmission timer and retransmits the lost packet with the lowest sequence number. The TCP sender resumes its timer when the corresponding ACK is received. The handling of ACK packets follows a similar process. The only difference is that the TCP sender will retransmit the highest ACK received so far. Besides that, EPLN does not use route discovery. The key idea here is to reduce the occurrences of route failures rather than to distinguish between congestion and wireless errors.

In sum, the aforementioned works have the following limitations. First, notifications or feedback from intermediate nodes may be lost, thus preventing the TCP sender from taking any actions. Secondly, they assume link failures are primarily caused by mobility. This is unrealistic since wireless links fade frequently due to interference and small scale fading [23]. Given that fades last for hundreds of milliseconds only, we argue that sending feedback to the sender unnecessarily degrade TCP's performance.

2.2 Wired-Wireless Networks

Besides WANETs, there have also been numerous works looking into improving TCP's performance over 1-hop wireless link. For completeness, we will review some relevant works before describing how we extended Snoop TCP to work in WANETs.

Balakrishnan et al. [2] propose the idea, called Snoop TCP, of caching packets and performing local recovery at the base station. The objective of Snoop TCP is to allow lost packets to be retransmitted without causing the TCP sender to invoke its congestion control or avoidance mechanisms unnecessarily.

In a different work, Arya et al. [1] describe an explicit mechanism that differentiates between wireless and congestion related losses. Their approach is based on a simple window framework where the window is used for storing information of a lost segment due to congestion or wireless errors. If a congestion or wireless loss is detected, the receiver notifies the sender, after which the sender adjust its sending rate.

Biaz et al. [4] propose using packets' inter-arrival times as the loss discriminator. Basically, when persistent congestion occurs, the congestion window is halved. In the case of wireless loss, TCP does not unnecessarily invoke its congestion control mechanism. However, their scheme assumes that the last link on a path is wireless and it's the

bottleneck link.

TCP Santa-Cruz [21] tries to differentiate congestion from random losses by monitoring the queue over a bottleneck link. TCP Santa-Cruz maintains a minimum queue size by adjusting the TCP's window size. From their research, they claim that congestion related losses can be identified when the queue length exceeds a certain threshold and wireless loss is not preceded by a queue build up event. When TCP Santa-Cruz detects wireless losses, no rate-halving is performed. However, when persistent congestion occurs, normal congestion ensues.

Finally, Garcia et al. [12] propose a novel approach that differentiates loss based on packet checksums. Their observation is that checksum errors are usually caused by wireless errors, thereby, a checksum error signifies wireless losses rather than congestion.

Table 1 summarizes the aboforementioned schemes. As mentioned, Snoop TCP has the nice feature of being transparent to both TCP sender and receiver. The main drawback, however, is that Snoop TCP has been designed to recover packets from a base station; i.e., it assumes a central node where packets can be recovered from. In contrast, WANETs consist of distributed nodes and packets are likely to traverse through multiple wireless links before reaching their destination. To this end, we propose InPCM (an extension of Snoop TCP) to facilitate the recovery of packets in multi-hop wireless networks.

3 In-Network Packet Caching Mechanism (InPCM)

InPCM's key idea is to have intermediate nodes use an in-network packet level caching mechanism to recover packets; similar in principle to Snoop TCP [2] but over WANETs.

Figure 1 gives an overview of InPCM which shows nodes B and C running InPCM, as illustrated by them caching packets destined to node E. Assume that a packet is lost on the link from node D to E, due to bad channel condition. Upon a timeout, node C retransmits the lost packet, thereby, preventing a timeout event at the sender. From the example above, we can draw a few observations. First, InPCM does not require every hop to perform caching. Second, InPCM recovers both data and ACK packets. Thirdly, InPCM can incorporate different caching and retransmission strategies that promote spatial reuse or avoid congested nodes. Finally, if node B or C moves away, one of them can take over.

3.1 Design and Implementation of InPCM

Conceptually, InPCM is similar to Snoop TCP, but the multi-hop nature of WANETs brings forth the following issues:

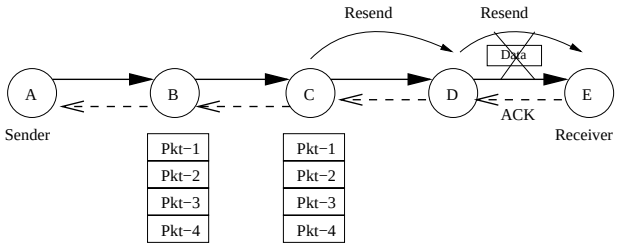

Figure 1. InPCM in operation.

- *Fine-grained timeout control*. We need to have a better control of the recovery process as injecting more packets unnecessarily into the network adds to the contention level.
- *Tracking incoming data and ACK packets*. Packet of different types belonging to different flows needs to be identified properly; otherwise, unnecessary packets will be cached.
- *Symmetric and asymmetric paths*. A link break causes the routing protocol to use an alternate route. As a result, it is conceivable that the new route may not contain intermediate nodes with previously cached packets.
- *Caching policies and cache size*. The policy used and number of hops to either the sender or receiver will dictate the required cache size.
- *Handling persistent congestion*. Naively retransmitting cached packets will exacerbate the network's congestion level, especially if all intermediate nodes retransmit the same packet.
- *Fairness among flows*. Given that different flows will experience different packet loss due to the wireless medium having location dependent error loss, we need to ensure a flow's poor link quality does not affect other flows' packet recovery.

An important issue highlighted above is when, how and what to cache. The size of the caches needs to be studied carefully and investigated so that a large number of flows can be accommodated. In the case where there are no constraints on cache sizes, the data cache will increase in proportional to an intermediate node's round trip time to the receiver and the number of flows. Furthermore, the location of a node affects the minimum required cache size, where a node closer to the destination will require less cache resources due to ACK packets taking less time to arrive. In this paper, we focus on computing the fine-grained

Table 1. Summary of TCP Enhancement Works.

Scheme	Type	Approach
TCP-F [5]	WANET	Alters TCP's state based on the received feedback.
ATCP [17]	WANET	Disables TCP's retransmission timer when a link is down and restores the timer after receiving the corresponding ACK for a data packet.
EPLN [28]	WANET	Disables TCP's retransmission timer when a link is down and restores the timer after receiving the corresponding ACK for a data packet.
Snoop-TCP [2]	WANET	Caches data and ACK packets, and using a fine grained timer to recover lost packets.
Arya et al. loss [1]	Wired-Wireless	Uses a simple window framework to differentiate packet loss and store loss segment information.
Biaz and Vaidya [4]	Wired-Wireless	Differentiate losses using inter-arrival times.
TCP Santa Cruz [21]	Wired-Wireless	Differentiate losses by monitoring queue over a bottleneck link.
Garcia and Brunstrom [12]	Wired-Wireless	Differentiate losses using checksum errors.

timeout, tracking of incoming data and ACK packets, and caching policies. Issues such as asymmetric paths, cache size and handling persistent congestion are our immediate future works.

3.2 System Overview

Figure 2 shows a system overview and key components necessary to realize InPCM. We design InPCM to sit between the MAC and network layer, thus giving it access to routing tables and close integration with the MAC. The role of each component shown in Figure 2 is discussed next.

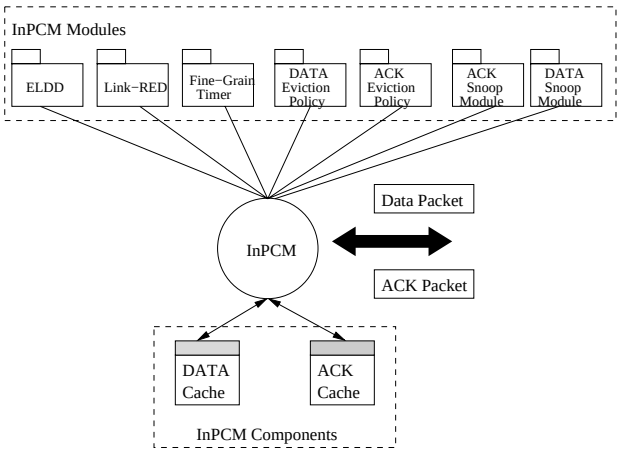

Figure 2. InPCM system overview.

3.3 Caching Algorithm

Figures 1 and 2 show how InPCM process data and ACK packets respectively. First, the processing of data packets is as follows. InPCM accesses the data packet's sequence number to validate whether the data packet is new, old, in-order or out-of-order. If the data packet is new (line-4), it is cached and forwarded to the next hop regardless of whether the data packet is in-order or out-of-order. Line 8 indicates that the receiver or some other nodes have not received the corresponding data packet, thus a retransmission of the last ACK packet is required. Line 9 is executed when either a data packet is lost or a new one arrives. For both cases, InPCM will cache the data packet and forward it onwards.

```
1  P ← Incoming Packet;
2  seqno ← GetSeqNo(P);
3  if P is Data-Packet then
4      if seqno == in_sync then
5          DataCache(P);
6          Forward(P);
7      end
8      else if seqno not in_sync then
9          if seqno ≤ Last_ACK.seqno then
10             Retransmit(Last_ACK);
11             Remove(P);
12         end
13         if seqno > Last_ACK.seqno then
14             DataCache(P);
15             Forward(P);
16         end
17     end
18 end
```

Algorithm 1: InPCM's Caching Algorithm: Data packet processing

ACK packets are processed in a similar manner. The ACK packet is first compared against the last ACK packet. Line 4 indicates the ACK packet is new. A node removes the corresponding data packet and update the last ACK seen. Line 11 handles the case where the sender timeouts or when nodes think that the sender has not received an ACK because of the delay in receiving a new data or ACK packet. InPCM does nothing in the former case but it will discard the packet in the latter. Further, when a duplicate ACK is received (line 14), it will check whether the corresponding data packet is in its data cache. If the data packet exists, the duplicate ACK is dropped.

Finally, InPCM maintains a counter that tracks the number of duplicate ACKs for each data packet. After receiving three duplicate ACKs, the corresponding cached data packet is retransmitted and the counter is reset. When an ACK packet arrives, InPCM evicts all data packets up to and including the acknowledged packet from its cache.

```
1  P ← Incoming Packet;
2  seqno ← GetSeqNo(P);
3  if P is ACK-Packet then
4      if seqno > Last_ACK.seqno then
5          RemoveCorrespondingDataPacket(P);
6          Remove(Last_ACK);
7          ACKCache(P);
8          Update(Last_ACK, P);
9          Update(Round Trip Time);
10     end
11     if seqno < Last_ACK.seqno then
12         Remove(P);
13     end
14     if seqno == Last_ACK.seqno then
15         if Data-Cache-Exist(P) then
16             Remove(P);
17             Increment(counter);
18             if counter==3 then
19                 RetransmitDataPacket(P);
20                 counter=0;
21             end
22             else
23                 Forward(P);
24             end
25         end
26     end
27 end
```

Algorithm 2: InPCM's Caching Algorithm: ACK processing.

3.4 InPCM's Eviction Policy

The main objective of evicting packets from the data and ACK cache is to purge outdated packets and make room for new packets. This is important considering that mobile nodes have limited resources; therefore, cache utilization must be kept at a minimum. A data packet will be discarded once its corresponding ACK arrives. In the case where TCP uses delayed acknowledgements, InPCM removes all data packets that have sequence numbers up to and including the acknowledged packet. Finally, an ACK packet will be removed from its ACK cache once a node receives a data packet with a higher sequence number.

3.5 Early Loss Detection and Discard (ELDD)

A data packet can be evicted early if a downstream node can guarantee that the data packet has been cached. The idea here is to eavesdrop on a neighboring node's transmission. Once the neighbor node transmits, a node can safely evict the transmitted packet from its cache. Note that this optimization means that a node is unable to enter power save mode until a neighbor transmits a cached data or ACK packet.

3.6 In-Network Recovery Timer

For each sender and receiver, an InPCM-enabled node maintains a fine-grained retransmission timeout (RTO) that is calculated based on its position relative to the sender and receiver. The reason for using a timer instead of waiting for three duplicate acknowledgements is because acknowledgement packets are susceptible to wireless related problems (e.g., high contention), thus ACKs cannot be relied on as a loss predictor. We now present how each node calculates its RTO value.

3.6.1 Standard Deviation-based Timer Mechanism

Initially, the InPCM's timer mechanism is based on how TCP calculates its RTO, referred subsequently as TCP-like. To see why the standard TCP mechanism cannot be used, consider the following analysis.

We set up a linear topology with varying number of hops and have intermediate nodes calculate RTO in a standard manner to the receiver. Here, we assume the packet's round trip time is constant. Constant round trip time means that the per-hop delay (phd) for the data and ACK packets is the same across multi-hop linear topology, although this might not be the case in the real world or simulation. However, this analysis is only to show that given a constant RTT, how different timer schemes are different in terms of the maximum retransmission. Further on, we analyze the number of retransmissions that can be attempted using the TCP-like timer mechanism and compare it against the maximum number of retransmission (MAX) in the ideal case. Here, we define the ideal case as one that provides the maximum retransmission attempts before the sender's RTO expires. MAX is calculated through ((4 X RTT) / RTT). The RTT of a node is calculated using (numberofhops X phd). The number of hops refers to the hops needed from a node to reach the sender including its way back.

Figure 4 shows that intermediate nodes using TCP-like lack retransmission opportunities when compared with the ideal case. This is undesirable if most of the packets are lost in the vicinity of downstream nodes and it is no different to having the TCP sender retransmit all packets.

Due to the above reasons, we propose another fine-grained timer based on RTT's standard deviation; its ability to grant more retransmission opportunities is clearly shown in Figure 4. To calculate a packet's timeout, InPCM uses the following equation.

$$ESRTO = SRTT + \beta \times (a + b \times hops) \qquad (1)$$

where ESRTO is the estimated RTO, $SRTT$ is the smooth RTT, β, a and b are constants, and $hops$ is number of hops to the receiver.

In words, the β value determines the aggressiveness of the retransmission process. Note, a more aggressive packet retransmission strategy does not necessarily lead to a better overall throughput performance, since more retransmissions will degrade performance due to increased contention level. Therefore, the β value must be chosen carefully so that all the nodes closer or farther away from the sender have a chance to perform retransmission and avoid overloading the network.

To select an appropriate β value, we conducted simulations using a static linear topology and vary the number of hops from 3 to 15. The simulation uses the parameters shown in Table 2. From the set of simulations performed, we found that a β value of 1.5 to be appropriate and achieves a stable performance over multi-hop simulations. We are in the process of investigating whether a suitable β value can be derived from the current contention level, the ACK's arrival time, and TCP's sender timeout, thereby, provides a generic scheme to determine β.

In our simulations, we found that the sampled RTT's standard deviation vary widely and does not increase linearly with hop count, see Figure 3. To "normalize" the standard deviation, we used a line fitting equation, Equ. 2, over the collected standard deviation values.

$$\overline{y} = a + b\overline{x} \qquad (2)$$

where a and b are regression coefficients, $\overline{y}$ is the estimated standard deviation, $\overline{x}$ is number of hops to the receiver, and n is number of samples being used. The regression coefficients of a and b are then obtained using Equ. 3 and 4. Finally, the resulting a and b values are inserted into Equ 1.

$$a = \frac{\overline{y}\left(\sum_{i=1}^{n} x_i^2\right) - \overline{x}\left(\sum_{i=1}^{n} x_i y_i\right)}{\sum_{i=1}^{n} x_i^2 - n\overline{x}^2} \qquad (3)$$

$$b = \frac{\left(\sum_{i=1}^{n} x_i y_i\right) - n\overline{xy}}{\sum_{i=1}^{n} x_i^2 - n\overline{x}^2} \qquad (4)$$

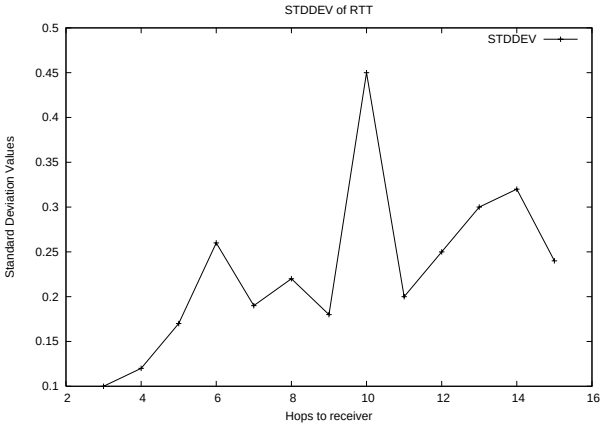

Figure 3. Standard Deviation of RTT.

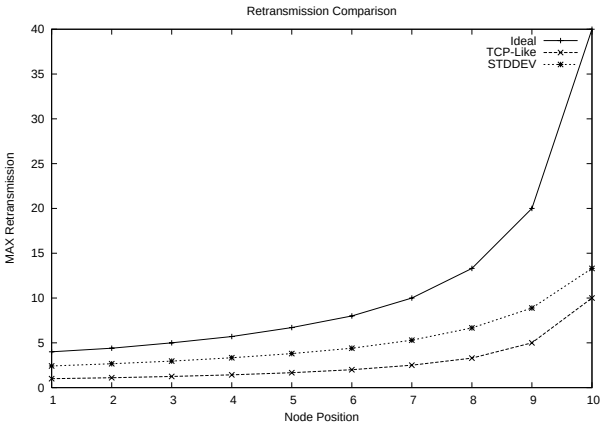

Figure 4. Comparison of maximum transmission under different timer mechanisms. The receiver is located after node-10.

3.7 Expired Data Packets Retransmission Policy

When an InPCM node experiences timeout, a naive way is to retransmit all expired data packets. However, having consecutive InPCM nodes retransmit the same packet unnecessarily increases the network's load.

The solution to this issue is to retransmit only the first unacknowledged data packet. To achieve this, we require the node whose retransmission fires first notifies upstream nodes to cancel all unnecessary retransmissions by sending a notification packet back to the TCP sender. The notification packet is only generated and processed by InPCM nodes. The InPCM node that generates this packet attaches information of the latest retransmitted packet types along with its sequence number.

An InPCM node that receives this packet scan through

its data packet and halts the corresponding packet's retransmission timer and evicts the packet from its cache. As an optimization, nodes may double their RTO value instead of evicting the data packet from its cache. This will help recover packets when nodes are mobile since this optimization increases the number of nodes which a receiver can rely on to recover packets.

3.8 Expired ACK Packets Retransmission Policy

Upon receiving new ACKs, an InPCM node will evict old ACK packets from its ACK cache. When an InPCM node does not receive any new data packets after some time, it will resend the latest acknowledgement packet. Bare in mind that not receiving any data packet could be due to the TCP sender not having any more data to send. To cater for this case, InPCM retransmits an ACK packet a maximum of three times before giving up, at which point the ACK packet is discarded.

3.9 Reducing Contention Level

TCP has been shown to increase contention in WANETs, see [11]. A problem with InPCM is that due to its retransmission mechanism, it too will add to the contention level. As a result, we need to carefully control the contention level in WANETs.

We address this problem by incorporating the Link-RED [11, 10] mechanism into InPCM. The idea here is to reduce TCP's aggressiveness so that additional bandwidth are available to perform retransmissions, thus keeping the contention level equal or less than not running Link-RED, but with the added performance gains provided by InPCM.

4 Simulation

We verify the performance of InPCM using *ns-2* (v2.28) [18]. Unless specified otherwise, all the simulations use the parameters shown in Table 2.

Our experiments consist of three network topologies: random, linear and grid. Figures 5 and 6 show an example of linear and grid topology respectively. The reason for using a linear topology is that it provides a controlled environment that simplifies the analysis of InPCM. The grid topology provides a more challenging scenario where we tested InPCM against frequent route changes, contention among flows and asymmetric paths. Finally, the random topology serves as a more realistic topology for testing InPCM where nodes experience frequent route changes due to mobility or contention among flows.

In our experiments, we collected the following metrics:

Table 2. Simulation Configurations.

Type	Used
MAC Protocol	IEEE 802.11
TCP Flavor	TCP Reno
Radio Propagation	2-ray Ground
Routing Protocol	DSR
Simulation Duration	1000s
Queue	CMUPriqueue
Packet Error Rate (PER)	2% and 5%
Node transmission range	200 meters
Simulation Runs	10
Topology Size	1000x1000 meters
Mobility Model	Random Way Point
Node speed	20 m/s

Figure 5. Linear topology.

- MAX_ACK. The largest ACK sequence number received by the sender. This metric measures the performance improvement without taking into account retransmitted packets (e.g., throughput). As a result, the measurement is more accurate as compared with throughput measurement.
- Data transfer time.
- Ratio of amount of time spent in the slow-start and congestion avoidance phases.

For the grid topology, see Figure 6, there are four TCP connections with flows emanating from node 1, 2, 3 and 4 to nodes 13, 14, 15 and 16 respectively. Furthermore, all the nodes are static and we used a PER of 5%. In the randomized topology, there are 20 nodes scattered in a 1000x1000m grid size.

5 Results

Figures 7 and 8 show the simulation results for the linear topology. In Figure 7, the objective is to show InPCM's performance gain compared to TCP without PER. We incorporated the Link-RED [11, 10] mechanism, as discussed in Section 3.9. In our simulations, we observe that the Link-RED mechanism is capable of lowering the contention level where a flow experiences fewer packet drops. We found that the combination of InPCM and LinkRED improves TCP's performance especially when contention level increases.

Figure 9 compares how TCP reacts to different PERs and how effective InPCM is in performing recovery for a given PER. We see that TCP treats the PER as though the network is congested. At a PER of 5%, TCP performs badly, registering little throughput. However, InPCM is able to perform recovery and hides most of the wireless losses from the TCP

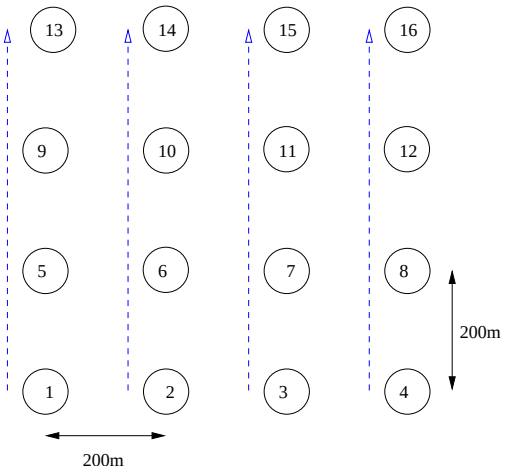

Figure 6. Grid topology.

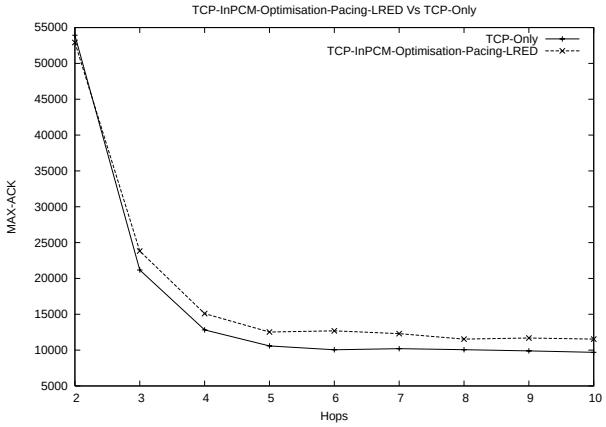

Figure 7. Performance of TCP vs InPCM (ELDD + Link-RED) during simulations without PER.

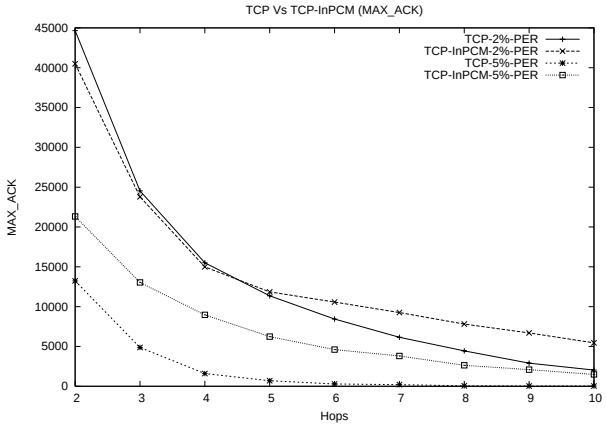

Figure 8. Performance of TCP vs InPCM with 2% and 5% of PER

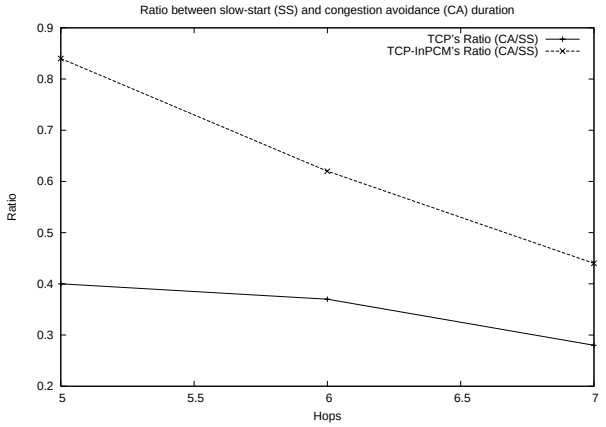

Figure 9. Ratio between slow-start and congestion avoidance (TCP vs InPCM)

sender. As a result, due to InPCM, the sender continues to send without invoking congestion control.

To better illustrate the effectiveness of InPCM, we consider the time spent in the congestion avoidance and slow-start phases. Again, we use a linear topology but with a PER of 5%. In Figure 9, we see that InPCM spends more time in the congestion avoidance rather than the slow-start phase. This is in contrast to not using InPCM which sees TCP spending the majority of its time in the slow-start phase.

The increase in TCP's performance leads to a reduced latency perceived by the end user. To verify this claim, we perform a simulation using a six hops linear topology with 5% PER, and a flow transferring 1 Mb of data. As shown in Figure 10, InPCM needs shorter amount of time than TCP to complete the data transfer.

Next, we conducted a set of simulation using a 4x4 grid topology. In this topology, InPCM is tested against frequent route changes due to contention and PER. Figure 11 shows that InPCM is able to improve TCP's performance (e.g., MAX_ACK) by 53% in aggregate even in the presence of frequent route changes and varying PERs. The reason is because InPCM sustains TCP flows in the congestion avoidance phase longer. Without InPCM, these flows timeout and enter slow-start frequently.

The last simulation uses the randomized topology with nodes placed in a grid of 1000 x 1000m with links having a PER of 5%. As shown in Figure 12, InPCM is able to improve the performance of TCP by 17% in the presence of mobility, frequent route changes and high PER. Notice that the second flow does not show any improvement. We believe if the fairness issue, discussed in Section 3.1, is addressed, then all InPCM flows will consistently outperform

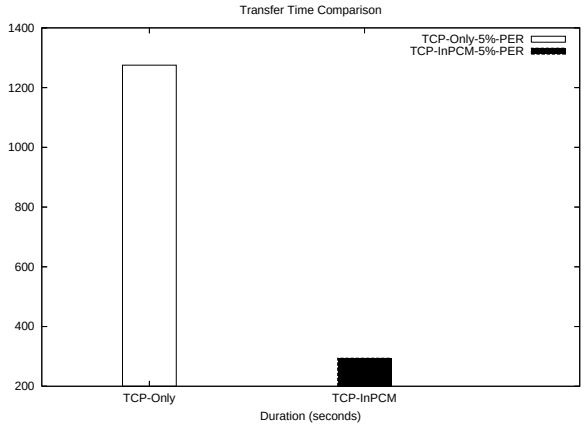

Figure 10. Perceived latency by TCP's sender in a data transfer.

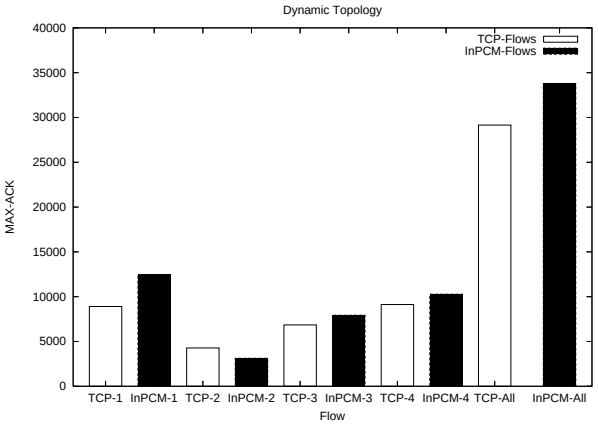

Figure 12. Dynamic topology simulation results.

other TCP flows.

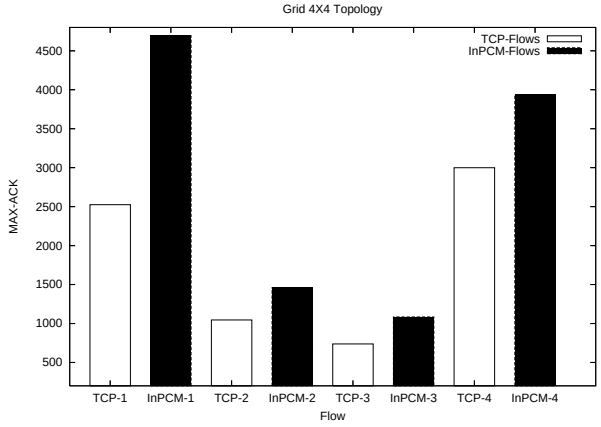

Figure 11. Grid (4x4) Simulation (TCP vs InPCM)

6 Conclusion

In this paper, we have proposed a novel caching mechanism called InPCM to address TCP's poor performance in WANETs. InPCM is an extension of the well-known Snoop TCP to multi-hop wireless networks. InPCM effectively augments the underlying MAC retransmission mechanism and is also able to incorporate mechanism such as Link-RED to reduce contention level. We have evaluated InPCM through simulation studies in various scenarios and our results indicate that InPCM is able to improve TCP's performance significantly. Some important areas requiring further work include fairness among flows, cache optimization and avoiding true congestion.

References

[1] V. Arya and T. Turletti. Accurate and Explicit Differentiation of Wireless and Congestion Losses. In *Proceedings of 23rd International Conference on Distributed Computing Systems Workshops*, pages 877–882. IEEE, 2003.

[2] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz. Improving TCP/IP Performance over Wireless Networks. In *Proceedings of the 1st Annual International Conference on Mobile Computing and Networking*, pages 2–11, New York, NY, USA, 1995.

[3] D. Berger, Z. Ye, P. Sinha, S. Krishnamurthy, M. Faloutsos, and S. K. Tripathi. TCP Friendly Medium Access Control for Ad hoc Wireless Networks: Alleviating Self Contention. In *1st IEEE International Conference on Mobile Ad hoc and Sensor Systems*, pages 214–223, Lauderdale, FL, 2004.

[4] S. Biaz and N. Vaidya. Discriminating Congestion Losses from Wireless Losses using Inter-Arrival Times at the Receiver. In *Proceedings of Application Specific Systems and Software Engineering and Technology*, pages 10–17, Richardson, TX, 1999. IEEE.

[5] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash. A Feedback-based Scheme for Improving TCP Performance in Ad hoc Wireless Networks. *Personal Communications, IEEE*, 8(1):34–39, 2001.

[6] K. Chen, Y. Xue, and K. Nahrstedt. On Setting TCP's Congestion Window Limit in Mobile Ad hoc Net-

works. In *IEEE International Conference on Communications*, pages 1080–1084, 2003.

[7] X. Chen, H. Zhai, J. Wang, and Y. Fang. TCP Performance over Mobile Ad hoc Networks. *Canadian Journal of Electrical and Computer Engineering*, 29(1):129–134, 2004.

[8] J. Cho, S. Oh, J. Kim, H. H. Lee, and J. Lee. Neighbor Caching in Multi-hop Wireless Ad hoc Networks. *Communications Letters, IEEE*, 7(11):525–527, 2003.

[9] M. Corp. Mobile Mesh Networks Technology. Technical Report. http://www.motorola.com/.

[10] Z. Fu, P. Zerfos, H. Luo, L. Zhang, and M. Gerla. The Impact of Multihop Wireless Channel on TCP Performance. *IEEE Transactions on Mobile Computing*, 4(2):209–221, 2005.

[11] Z. Fu, P. Zerfos, H. Luo, L. Zhang, S. Lu, and M. Gerla. The Impact of Multihop Wireless Channel on TCP Throughput and Loss. In *22nd Annual Joint Conference on the IEEE Computer and Communications Societies*, pages 1744–1753, 2003.

[12] J. Garcia and A. Brunstrom. Transport Layer Loss Differentiation and Loss Notification. First Swedish National Computer Networking Workshop.

[13] M. Gunes and D. Vlahovic. The Performance of the TCP/RCWE Enhancement for Ad hoc Networks. In *Proceedings of 7th International Symposium on Computers and Communications*, pages 43–48, 2002.

[14] Y. He, C. S. Raghavendra, S. Berson, and R. Braden. TCP Performance with Active Dynamic Source Routing for Ad hoc Networks. In *Proceedings of International Workshop on Active Network Technologies and Applications*, pages 24–35, Osaka, Japan, 2003.

[15] X. Hou and D. Tipper. Impact of Failures on Routing in Mobile Ad hoc Networks using DSR. In *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference*, Orlando, FL, 2003.

[16] H. Lim, K. Xu, and M. Gerla. TCP Performance over Multipath Routing in Mobile Ad hoc Networks. In *IEEE International Conference on Communications*, pages 1064–1068, 2003.

[17] J. Liu and S. Singh. ATCP: TCP for Mobile Ad hoc Networks. *IEEE Journal on Selected Areas in Communications*, 19(7):1300–1315, 2001.

[18] S. McCanne and S. Floyd. ns Network Simulator-2. http://www.isi.edu/nsname/ns/.

[19] J. P. Monks, P. Sinha, and V. Bharghavan. Limitations of TCP-ELFN for Ad hoc Networks. In *Proceedings of the 7th International Workshop on Mobile Multimedia Communications*, Marina del Rey, CA, 2000.

[20] K. Nahm, A. Helmy, and C. C. J. Kuo. TCP over Multihop 802.11 Networks: Issues and Performance Enhancement. In *Proceedings of the 6th ACM International Symposium on Mobile Ad hoc Networking and Computing*, pages 277–287, Urbana-Champaign, IL, USA, 2000.

[21] C. Parsa and J. J. G. L. Aceves. Differentiating Congestion vs Random Loss: A Method for Improving TCP Performance over Wireless Links. In *Wireless Communications and Networking Conference*, pages 90–93, Chicago, IL, 2000. IEEE.

[22] J. Postel. RFC 793 - Transmission Control Protocol, 1981.

[23] T. S. Rappaport. *Wireless Communications: Principles and Practice*. Prentice-Hall, 1996.

[24] M. Stangel and V. Bharghavan. Improving TCP Performance in Mobile Computing Environments. In *IEEE International Conference on Communications*, pages 584–589, Atlanta, GA, 1998.

[25] K. Sundaresan, V. Anantharaman, H.-Y. Hsieh, and R. Sivakumar. ATP: A Reliable Transport Protocol for Ad hoc Networks. In *Proceedings of the 4th ACM International Symposium on Mobile Ad hoc Networking and Computing*, pages 64–75, Annapolis, Maryland, USA, 2003.

[26] K. Tang and M. Gerla. MAC Reliable Broadcast in Ad hoc Networks. In *Military Communications Conference*, pages 1008–1013, 2001.

[27] K. Xu, S. Bae, S. Lee, and M. Gerla. TCP Behaviour across Multihop Wireless Networks and the Wired Internet. In *Proceedings of the 5th ACM International Workshop on Wireless Mobile Multimedia*, pages 41–48, Atlanta, Georgia, USA, 2002.

[28] X. Yu. Improving TCP Performance over Mobile Ad hoc Networks by Exploiting Cross-Layer Information Awareness. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, pages 231–244, Philadelphia, PA, USA, 2004.