2013-09-27

JTP, an energy-aware transport protocol for mobile ad hoc networks (PhD thesis)

BOSTON UNIVERSITY
GRADUATE SCHOOL OF ARTS AND SCIENCES

Dissertation

JTP, AN ENERGY-AWARE TRANSPORT PROTOCOL FOR MOBILE AD HOC NETWORKS

by

NIKY RIGA
MA, Boston University, 2007

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
2013

Approved by

First Reader

Abraham Matta, Ph.D.,
Professor of Computer Science
Boston University

Second Reader

Craig Partridge, Ph.D.,
Chief Scientist
Raytheon BBN Technologies

Third Reader

Azer Bestavros, Ph.D.,
Professor of Computer Science
Boston University

DEDICATION

To my friends, that bring light to my life even in its darkest hours.

Acknowledgments

The PhD journey is different for everyone but always full of experiences and lessons. The biggest lesson for me was that I can not do everything alone. This would not have been a successful journey if it has not been for the help of many different people that gave me the support I needed in order to finish what I started many years ago.

First of all I would like to thank my advisor Abraham Matta, for his patience and guidance throughout the years. He gave me the freedom to find my own path and was always there to help and advise me. I would also like to thank my co-advisor Craig Partridge that taught me how to properly document and found my work. He also showed me how to leverage prior research to expand my knowledge and understanding of a research field. Both of my advisors are great teachers and I was very fortunate to work with them. The rest of my committee – Azer, Jason, John and Mark – also provided invaluable feedback; thank you all very much.

This thesis would have not been possible without the help of the JAVeLEN team. I want to thank Alberto for being my mentor while designing JTP, Dan and Will for accommodating all my crazy requests to support and implement my ideas and above all Jason that gave me the opportunity and freedom to innovate, and trusted me to develop this work within the JAVeLEN system.

During these years there have been many people that ensured I do not give up and helped me to march along even when it seemed that this will be an endless journey. Specifically, I would like to thank Tim and Stan for always motivating me to continue.

Through the years of working at BBN, I was very fortunate to work with people that showed great understanding and helped me find the time to work on my thesis. Thank you

Vikas, Mark, Vic and Sarah.

I want to thank my friends that supported me through the most difficult times and were there whenever I needed them. I want to specially thank Angela for being my Skype champion and for always being available to convince me that this is a fight worth fighting, Dido that was my thesis buddy and helped me get through the writing of most of this thesis and finally Stavros and Panos who helped me keep my sanity over the years. In no particular order I would also like to thank: Matina, George, Yola, Evimaria, Thomas, Panagiotis, Aristeidis. This is an incomplete list and it is impossible to properly credit all the people that have helped me through this journey, I just hope that I have already shown my gratitude to them.

Last but not least I would like to thank my family for all their love and support.

JTP, AN ENERGY-AWARE TRANSPORT PROTOCOL FOR MOBILE AD HOC NETWORKS

(Order No.)

NIKY RIGA

Boston University, Graduate School of Arts and Sciences, 2013

Major Advisor: Abraham Matta, Professor of Computer Science,

ABSTRACT

Wireless ad-hoc networks are based on a cooperative communication model, where all nodes not only generate traffic but also help to route traffic from other nodes to its final destination. In such an environment where there is no infrastructure support the lifetime of the network is tightly coupled with the lifetime of individual nodes. Most of the devices that form such networks are battery-operated, and thus it becomes important to conserve energy so as to maximize the lifetime of a node.

In this thesis, we present JTP, a new energy-aware transport protocol, whose goal is to reduce power consumption without compromising delivery requirements of applications. JTP has been implemented within the JAVeLEN system. JAVeLEN [RKM+08], is a new system architecture for ad hoc networks that has been developed to elevate energy efficiency as a first-class optimization metric at all protocol layers, from physical to transport. Thus, energy gains obtained in one layer would not be offset by incompatibilities and/or inefficiencies in other layers.

To meet its goal of energy efficiency, JTP (1) contains mechanisms to balance end-to-end vs. local retransmissions; (2) minimizes acknowledgment traffic using receiver regulated rate-based flow control combined with selected acknowledgments and in-network caching of packets; and (3) aggressively seeks to avoid any congestion-based packet loss. Within this ultra low-power multi-hop wireless network system, simulations and experimental results demonstrate that our transport protocol meets its goal of preserving the energy efficiency of the underlying network. JTP has been implemented on the actual JAVeLEN nodes and its benefits have been demonstrated on a real system.

Contents

List of Tables

List of Figures

List of Abbreviations

ACK	.....	ACKnowledgement
ADU	.....	Application Data Unit
AIMD	.....	Additive Increase Multiplicative Decrease
ALF	.....	Application Level Framework
API	.....	Application Programming Interface
ARL	.....	Army Research Lab
ATM	.....	Asynchronous Transfer Mode
ATC	.....	Application Transfer Controller
ARPANET	.....	Advanced Research Projects Agency NETwork
ARQ	.....	Advanced Research Projects Agency NETwork
BER	.....	Bit Error Rate
CE	.....	Connection Establishment
CE	.....	Connection Reset
CT	.....	Connection Termination
CSMA/CA	.....	Carrier Sense Multiple Access/Collision Avoidance
CPU	.....	Central Process Unit
DSP	.....	Digital Signal Processor
DPS	.....	Dynamic Packet State
E2E	.....	End-to-end
eJTP	.....	end-to-end JTP
EWMA	.....	Expontially-Weighted Moving Average
FEC	.....	Forward Error Correction
FIFO	.....	First Info First Out
GPS	.....	Global Positioning Sytem
GSM	.....	Global Sytem for Mobile

iJTP	.....	intermediate JTP
IP	.....	Internet Protocol
ITU	.....	International Telecommunication Union
JAVeLEN	.....	Join Architecture Vision for Low Energy Networking
JAVRoute	.....	JAVeLEN Routing
JNC	.....	JTP No Caching
JTP	.....	JAVeLEN Transport Protocol
LRU	.....	Least Recentrly Used
LSU	.....	Link State Update
LT	.....	Loss Tolerance
MAC	.....	Medium ACcess
MANET	.....	Mobile Ad Hoc NETwork
MOS	.....	Mean Opinion Score
MRU	.....	Most Recentrly Used
OLSR	.....	Optimized Link State Routing
OS	.....	Operating System
OSAL	.....	Operating System Abstracion Layer
P2P	.....	Point-to-point
PER	.....	Packet Error Rate
PESQ	.....	Perceptual Evaluation of Speech Quality
PLF	.....	Portable Link Framework
PI	.....	Proportional Integral
PI^2/MD	.....	Proportional Integral/Multiplicative Decrease
PIR	.....	Passive InfraRed
PSTN	.....	Public Switched Telephone Networks
RLE	.....	Run Length Encoding
RSSI	.....	Received Signal Strength Indication
RFC	.....	Request For Comments
RTO	.....	Retransmission TimeOut
RTT	.....	Round Trip Time
QoS	.....	Quality of Service
SACK	.....	Selective ACKnowledgement
SNACK	.....	Selective Negative ACKnowledgement
SNR	.....	Signal to Noise Ratio
TCP	.....	Transport Control Protocol
TOC	.....	Tactical Operations Center
TTL	.....	Time To Live
TDMA	.....	Time Division Multiple Access
UDP	.....	User Datagram Protocol
VoIP	.....	Voice over IP
WWP	.....	Wave and Wait Protocol
Xmit	.....	Transmit
XCP	.....	eXplicit Control Protocol
XTP	.....	eXpress Transport Protocol

Chapter 1

Introduction

1.1 Motivation

Ad hoc networks are self configuring networks of devices connected with wireless links. A node that is a member of such a network relies on wireless communication in order to find neighboring nodes, determine paths, and ultimately forward packets over multiple hops from a source to a destination, all without the use of fixed infrastructure. Ad hoc networking is therefore of most value in places where wireless communication is necessary, and infrastructure is not available, cannot be trusted, or is too expensive [MK05]. Moreover, its decentralized nature makes ad hoc networking a promising technology for large-scale deployments.

Situations where ad hoc networks are suitable often imply scenarios where nodes are required to operate on limited energy sources for extended periods of time; wireless sensor systems are an obvious setting where energy is clearly a concern [MK10]. Other systems include networks of robot teams, networks of emergency personnel, or soldier radios.

Apart from the challenge of energy limits, ad hoc networks inherit all the vices of wireless networks. Unlike their wire-line counterparts, wireless networks are plagued with unique difficulties such as contention for the wireless medium, unpredictable link quality, and time-varying topology. Researchers have studied these problems extensively [Per01].

In recent years researchers have started focusing more on the energy efficiency of such systems since most of the ad hoc application's service life – and thus usability– directly depends on the lifetime of nodes. Reducing the energy consumed in the radio circuit is of significant concern since it draws a lot of the battery power.

Over the years hardware and protocol designers have tried to address this challenge in a piecemeal fashion [Tor, YZJ04, SBS02, YGE01, YF04, RK01, SR02, BRBR03b, SR98, SRS01]. The problem with such piecemeal approaches is that when these mechanisms and protocols are put together to operate in a single environment it is likely that contradicting design choices, as well as inefficiencies in some layers, will offset the gains obtained by others. This is especially true when the goal is to minimize the total energy spent in the system, where any inefficiency translates in extra joules. Good examples are the extra headers that each layer appends to the packet, often carrying duplicate information, or the control packets that each layer injects into the network in isolation. Much of this overhead can be eliminated by cooperation and coordination. A good example of redundancy is the information in the headers for uniquely identifying nodes. In wired networks there is a need for both the link layer and the routing layer to include separate node identifiers. However, in Mobile Ad hoc networks many times the nodes are assigned unique IDs and there is no need for duplicating this information in the headers of both layers. On the other hand collapsing all layers and creating a monolithic system has its own well-known problems – too application specific, not extensible and not reusable.

In this thesis we examine the problem of designing an energy-conserving transport protocol. However instead of trying to solve this problem in isolation, we developed our protocol, JTP, within JAVeLEN, an energy-conscious system. JAVeLEN (Joint Architecture Vision for Low Energy Networking) [RKM+08] – a DARPA funded deployment – uses a complete system approach with an architecture design targeting energy efficiency. Although JAVeLEN maintains the layering in protocol design, it encompasses a set of inter-operating mechanisms that collectively aim to significantly reduce the energy spent under varying network sizes, traffic rates, mobility patterns, and network densities. JAVeLEN achieves

dramatic results demonstrating networks that consume 100 times less energy for the same effective network goodput, compared to a typical 802.11 multi-hop wireless network running the OLSR [CJ03] routing protocol. Although JAVeLEN is designed for mobility, it specifically target networks with low-offered load and moderate mobility – mobility corresponding to running speeds.

JTP is designed to maximize its energy gains by leveraging the optimizations of the JAVeLEN system. We started our pursuit of designing a transport protocol for energy-conscious systems by reviewing the current literature. Extensive studies (reviewed in Chapter 2) have demonstrated the inadequacy of TCP [oSC81] to serve as a transport protocol in wireless environments. Later attempts to enhance or redesign TCP for wireless scenarios are mainly focused on improving performance in terms of goodput and delay, but not of energy consumption.

Although the JTP protocol described in this thesis was motivated by the real world problem of transporting application data within the aforementioned JAVeLEN system, its design is based on broad energy saving techniques for ad hoc and sensor networks that provide insight into how to save energy at the transport layer. JTP separates policies from mechanisms and thus can be adjusted to work within other architectures as well, as we will describe in Chapter 3.

To the best of our knowledge, there is no prior work on designing a generic transport protocol that seeks to achieve a higher network-wide energy efficiency by exploiting energy-gain opportunities from other layers. As we present in this thesis, the proposed protocol has a novel design whose goal is to minimize energy expended by the whole system while still satisfying the requirements of the application. Although JTP can improve energy consumption within any MANET, it achieves the greatest improvement within the settings that JAVeLEN is envisioned to operate; i.e. in networks with low-offered low and moderate mobility.

1.2 Summary and Contributions

JTP is an energy-efficient transport protocol that is implemented within the JAVeLEN system (see Chapter 2 for more details on the system). JTP is responsible for data exchange between applications in the network. In this role, JTP mediates between an application's need to share information of varying importance, and JAVeLEN's goal of minimizing energy expenditure per successfully delivered bit. Although JTP is implemented to work within JAVeLEN its design enables deployment within any wireless (or wire-line) architecture that provides an interface for JTP to both control the number of node retransmissions made by the media-access (MAC) protocol, and get an indication of the available bandwidth of the channel and of the packet loss rate.

The contributions span three aspects: (1) the architectural design of JTP and where functionality is placed within the network stack; (2) the design of JTP itself and how it separates mechanism from policy; and (3) the performance analysis and evaluation of JTP.

JTP's architecture strives to maximize the modularity of the system. Information is propagated through packet headers and all the necessary hop-by-hop operations are performed by a separate module that is used by the MAC.

A principle that aids the coherence of JTP's design, along with simplicity and reusability, is that of separating mechanism from policy [Tan02]. To this end, JTP contains a set of mechanisms for connection management, packetization, flow control, error control, service quality control, in-network caching, and path quality assessment. Policies are isolated so they could be exchanged without affecting the mechanisms. Throughout we describe such mechanisms, along with specific policies implemented in the current version and evaluated in this thesis. The goal behind our choices is to save energy while satisfying a range of reliability levels.

JTP employs mechanisms akin to the Dynamic Packet State [SZ99]—using packet headers to propagate information—to avoid maintaining per-flow state in intermediate nodes and maintain the modularity of the system. To the best of our knowledge JTP is the first

multi-hop wireless *end-to-end* transport protocol designed to perform hop-by-hop *soft-state* operations to improve goodput and energy performance while preserving the *end-to-end principle* [SRC81] (Chapter 3).

JTP exploits any energy-gain opportunities provided by the applications. Historically, transport protocols have offered a particular reliability/QoS model and the application's task was to pick the transport protocol whose model most closely met the application's needs (e.g. UDP, TCP, ITP [RBS00], RTP [SCFJ96]).

JTP is designed to serve as a generic transport protocol tailored by the application for its specific QoS semantics, which to the best of our knowledge has not been implemented before (see Chapter 4). JTP uses the tolerance of the application to packet loss to limit the network's effort in delivering a packet based on the packet's individual importance as well as current energy costs (Chapter 5).

As noted also in NETBLT [DDC87], the receiver has more information about the data transfer than the sender. In JTP, building upon prior work we take this one step further and make the receiver fully responsible for controlling all transmission parameters – connection's sending rate, retransmission requests for missing/lost packets, as well as the frequency of such controls. To the best of our knowledge, JTP is the first transport protocol that supports *variable destination-controlled feedback* trying to keep feedback as low as the stability and reliability of the network permits (Chapter 6).

Ludwig, in his thesis [Lud00a], thoroughly describes the shortcomings of end-to-end error control. To address this, JTP implements a caching mechanism (see Chapter 5) which enables intermediate nodes along the path of a JTP connection to temporarily store traversing packets. Efficiency achieved by in-network caching for repairing errors sooner does not contradict the end-to-end argument of system design [SRC81]—the source does *not* delete its copy of a packet until it gets an acknowledgment from the final destination, and the source is still responsible for ensuring the integrity of the transfer. Furthermore, the soft-state nature of caches provides resilience to route changes. These pipelines of caches along paths generalize the single-level caching often employed in cellular-type (single wireless

hop) networks [BSAK95]. Although a system that supports symmetric routes between hosts, like the JAVeLEN system, would exploit caching benefits to its fullest, the opportunistic design of this caching system seizes any chance for locally recovering lost packets, without interfering with the end-to-end semantics of each connection.

To allocate bandwidth fairly among flows in the presence of in-network caching and retransmissions a JTP sender backs off its sending rate to account for "internal" retransmissions triggered from caches on its behalf (Chapter 5).

JTP also employs a congestion-avoidance, rate-based flow control, using ATM-like explicit rate feedback from the network, in an attempt to eliminate energy waste associated with congestion-induced packet drops.

JTP is implemented as shared code that can be run either in simulation or on real radio nodes. Results from simulation and from a prototype of JAVeLEN radios (Chapter 7) confirm the premise of JTP in reducing the energy consumed per delivered bit.

The rest of the thesis is organized as follows: Chapter 2 provides a summary of the JAVeLEN system and discusses other works that are relevant to JTP. Chapter 3 presents the JTP architecture, introduces the various interfaces and describes the packet formats. Chapter 4 describes in detail the mechanisms that enable JTP to adapt to different application requirements. Chapter 5 discusses mechanisms that JTP employs for error control and presents the policies that have been implemented. Chapter 6 presents JTP's mechanisms for flow control and it describes how JTP monitors the path and adapts to changes. Chapter 7 presents the implementation details as well as simulation and real life results. Finally the thesis concludes with Chapter 8.

Chapter 2

Background

Wireless ad hoc networks have attracted much attention as the technology of choice for large-scale distributed sensing and communication in environments where existing infrastructure is not available, cannot be trusted, or is too expensive for the particular application. Many of these implementations utilize battery powered nodes that limit their lifetime. When nodes run out of battery not only data are lost but also the network's sustainability is affected since in ad-hoc networks all nodes also act as routers ensuring the end-to-end communication between various hosts [ASSC02]. Thus, the network's uninterrupted service life is limited by the energy efficiency of each node. Although advances in the battery technology can help they are not expected to significantly extend the lifetime of the network, leading researchers to look into software and hardware methods for reducing total energy consumption for multi-hop wireless networks.

2.1 The JAVeLEN System

JAVeLEN is an ad hoc network system that is designed to be extremely energy efficient, particularly when stationary or under low offered loads, and yet it also supports node mobility and high data rates. In wireless nodes a significant consumer of energy is the radio idle listening, receiving and transmitting [GW02]. Although energy gains can be obtained by efficient management of other components (CPU, memory, etc.), unless the radio usage

is greatly reduced the battery life can't be extended significantly. This observation drives the main design of JAVeLEN, which strives to minimize all radio usage. JAVeLEN's goal is a departure from many existing sensor systems such as the Berkeley Motes [mic], which use very little energy but only support low data rates and little or no mobility. JAVeLEN's broader intention was to design a system that works in a wide range of situations while yielding extraordinary energy consumption in all of them.

JAVeLEN's approach for designing a highly energy-conserving system is network-centric and allows the efficient and proper use of advances in low power electronics. JAVeLEN's design is network-centric, in the sense that instead of trying to improve and design new energy-efficient hardware, it focuses on improving the network protocols, where a lot of the energy waste is coming from, and then influences the hardware design based on protocol requirements. JAVeLEN's design is also unique because it begins with a clean slate. The goal is not to maintain backward compatibility with IP networking, or to focus on a particular application. This choice is not unlike the way the original ARPANET designs were created unencumbered by prior applications, protocols, or hardware. To make the most of every joule available in very dense and very large sensor systems requires an aggressive set of ideas and capabilities. In order to save energy system-wide it is critical to primarily attack the energy demands of the part of the system that operates, views, and adapts at both the point-to-point and end-to-end levels of the network. JAVeLEN provides this new paradigm of network thinking and provides the roadmap for the overall system.

Recognizing the inefficiencies introduced by the layering approach, JAVeLEN's design aims to eliminate them, not by collapsing all layers, but by introducing a tighter collaboration between them. By allowing cross-layer interactions JAVeLEN enables upper layers to build upon energy efficiencies provided by the lower ones. Figure 2.1 shows the layering in the JAVeLEN system as well as a high-level sketch of the cross-layer interactions. Following is a more detailed description of the role of each layer other than the transport layer. The interactions between layers are also presented emphasizing the new capabilities offered to protocol designers at the higher layers.

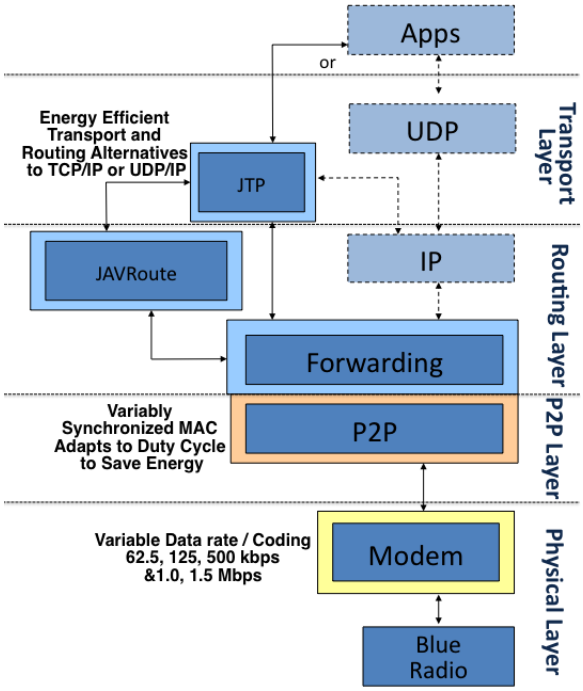


Figure 2.1: JAVeLEN network stack

2.1.1 Physical Layer

The **Physical Layer** is responsible for delivering bits between adjacent nodes. The **modem** module does the modulation and coding for all outgoing packets, as well as the demodulation and decoding for all incoming packets. The **transceiver** is responsible for sending/receiving the modulated information over the air. The transceiver for JAVeLEN is an energy-conserving radio which supports low-power sleep mode and fast switching times, achieving great energy savings while not compromising performance. The radio also encompasses the use of different physical waveforms – there is a low-data rate, energy optimized Hail waveform; and also some high-data rate waveforms optimized for short or long packets. JAVeLEN uses these different waveforms in a unique way, so as to maximize energy savings and ensure that the radio consumes a lot of power only when there are data to be sent [RKM+08]. The Hail waveform can be thought of as a door bell. When a transmitter has data to send, it first transmits a low-power signal of a specific pattern (*Hail*) to the

receiver (or, for multicast, receivers) with which the transmitter wishes to communicate. This pattern is recognized and the radio switches to the high-data rate waveform to receive the actual data packet. This ensures that the radio only uses substantial receive power when a sender has explicitly indicated that it has data to transmit.

The transceiver and the modem modules expose all these features to the Link Layer that is able to constantly fine tune the operating parameters, to optimize performance.

2.1.2 Point-to-Point (P2P) Layer

The **Point-to-Point (P2P) Layer** manages the radio in order to deliver packets between adjacent nodes, while minimizing the energy consumption. It also implements the radio frequency (RF) link characterization, based on which it performs link and neighbor establishment. The JAVeLEN MAC is a highly energy-efficient, slotted protocol (TDMA-like). To keep energy low, the radio is turned on only in a fraction of slots (wakeup slots). The percentage of wakeup slots is dynamically adjusted based on the actual network usage. Pseudo-random sequences are used, to allow a node to compactly express wakeup times that are uncorrelated with other nodes, yet completely predictable from slot to slot. Furthermore, the frequency of wakeup is expressed in the form of transmit and receive probabilities, and can easily be adapted on a per node basis given particular throughput and energy requirements.

Neighbor changes are determined through the use of periodic heartbeats. The heartbeat rate is dynamically adjusted based on the surrounding mobility, since in stationary networks the neighbor information is not expected to change frequently. Physical layer statistics (e.g. Received Signal Strength Indication(RSSI)), link characteristics (e.g. utilization) as well as neighbor information are monitored and reported to other layers. This information enables other layers in the networking stack to make more informed decisions about packet routing, transmission rates, etc.

Wireless communications experience much higher bit error rates, than their wire-line counterparts, causing packets from higher layers to get lost. This could result in inaccurate

information and retransmissions from higher layers, causing inefficient use of the channel and increasing the energy expenditure. All these could be avoided if the P2P module employs various mechanisms to increase the reliability of the channel. To this end, the JAVeLEN MAC:

- employs **Hail and Data Acks**. This avoids pointless transmissions, and ensures better feedback on data transmissions to other layers.

- employs **multiple link layer transmissions** of the same data packet. Many times the bit errors are not caused by a general bad condition of the channel, but are just random; some packets are just unlucky. In this case it is probably more beneficial for the MAC to retransmit the data packet, to avoid invoking loss recovery mechanisms from the higher layer. The idea of making the link layer more reliable was first deployed in X.25 [(IT96] networks. Moreover 802.11 uses multiple Automatic Repeat reQuests (ARQs) to increase the reliability of the channel. The JAVeLEN MAC however, supports dynamic number of retransmissions that can be set per packet.

- employs dynamic, per neighbor configuration of the transmission power for data packets. This allows energy conservation, as well as maximizes spatial reuse, without sacrificing reliability.

The above functionalities are not only deployed by the JAVeLEN MAC, but are also exposed to other layers, with the use of transmission profiles, so that the routing for example can decide how many times a Link State Update (LSU) should be retransmitted if it fails, and what power should the transceiver use. Each data transmission request that arrives to the JAVeLEN MAC, is accompanied by a transmission profile that specifies the transmission parameters for that packet. The JAVeLEN MAC also provides default values for these parameters that can be used, if desired.

The JAVeLEN MAC architecture also provides the ability for *Plugins* to be introduced into the JAVeLEN MAC architecture for two primary reasons:

1. There is a need to allow upper-layer, non-real-time protocols (e.g. routing, transport, etc.) and applications to insert functionality into the lower level real-time (MAC) processing path to support:

   - **Mid-hop transaction efficiency.** Higher level modules and applications occasionally have a need to perform operations at each mid-hop. It is more efficient to do this operation within the real-time mid-hop processing path, rather than to push the data through a protocol stack, do the operation and push it back down to the real-time protocols.
   - **Accessibility to MAC level information.** Some information is more easily available, or is only accurate at the MAC level. Higher level algorithms can be more flexible and efficient when this information is available.
   - **Just-in-time Packet Modification.** All MAC layers incur some delay on packet transmission, so it is helpful to be able to modify the packet and decide on whether it should be sent or not just before its transmission.

2. The plugin model gives the ability to experiment with new ideas without disrupting the rest of the system.

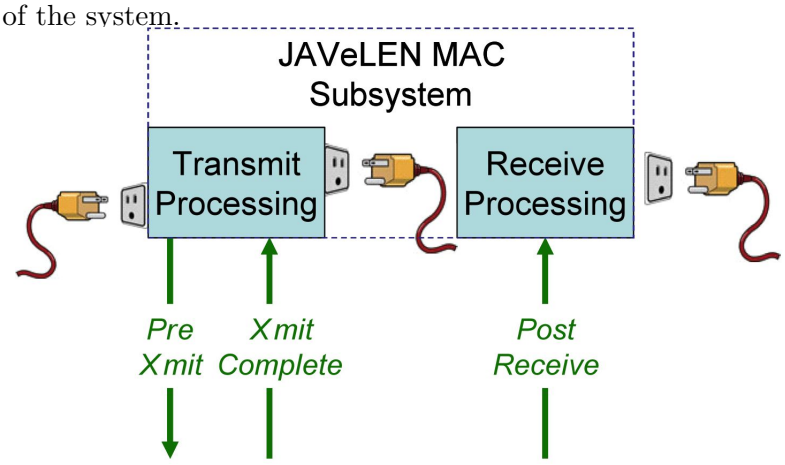


Figure 2.2: The packet flow in the JAVeLEN MAC through the plugins

The plugins have three major call points as shown in Figure 2.2 :

1. Right before a message is transmitted; this is useful for just-in-time packet updates.

2. After a transmission is complete; this is useful to know when the packet went out and whether the transmission was successful or not.

3. Right after a message reception; this is useful for handling mid-hop operations.

Except from the above packet-specific events, the plugins can also set individual timers and get called upon timer-expiration or when receive messages from the rest of the system.

The JAVeLEN Challenge

The JAVeLEN architecture has been developed to elevate energy efficiency as a first-class optimization metric at all protocol layers, from physical to transport [RKM+08]. Thus, energy gains obtained in one layer would not be offset by incompatibilities and/or inefficiencies in other layers.

As stated by the authors [RKM+08], the largest contribution to energy efficiency in the JAVeLEN system comes from intelligent and adaptive use of the multi-waveform radio transceiver by the P2P protocol. Each waveform has particular capabilities suited for different situations. In particular, the use of the Hail waveform to ensure that the radio only operates in the high-power, high-datarate mode when data will be in flight, using the Hail waveform for basic timing, along with the pseudorandom schedules of the nodes that dynamically adapt to network traffic, result in nearly two orders of magnitude in power efficiency.

Providing such an energy-efficient communication substrate poses a challenge to upper layers to be energy conscious and not be wasteful in their communication. Using such an energy conserving communication foundation to build a system, will be of no use if the higher layers are chatty and waste energy. In order for the whole system to provide high energy savings all layers have to work in tandem, and aim to minimize communications.

2.1.3 Routing Layer

The **Path Management (routing) layer** is responsible for discovering the network topology and determining routes between hosts so that it can provide the next hop for each packet based on the final destination.

It is challenging to design a routing protocol that is parsimonious in its use of control traffic. In JAVeLEN energy gain mainly comes from keeping the radios off so it is desirable that control protocols do not transmit unnecessarily. At the same time, it is essential that connectivity among nodes is maintained, even when the nodes are mobile.

The JAVeLEN Routing (JavRoute) module uses link state information from nodes throughout the network to plan a path between source and destination. JavRoute is the only module in the system that maintains network-wide information, and it can provide this information to other modules. JavRoute encompasses the use of Hazy-sighted scoping [SRS01] to control the dissemination of routing information. It further employs energy conserving, multi-point relaying [JLMV02] using knowledge of transmission power at nodes to build a connected dominating set for route information dissemination. This combination gives the JAVeLEN system another order of magnitude in improvement in static networks while sending only a little more control traffic than OLSR in cases of mobility - all the while maintaining better connectivity over substantially more nodes.

Route generation is energy-sensitive, and battery-aware link-biasing is employed to send packets along the paths that require the minimum amount of power for their delivery.

The **Forwarding** module provides next hops to the P2P module based on the routes that JavRoute calculates.

2.1.4 Transport Layer

The **transport layer** is responsible for delivering data between end-host applications.While the optimization of energy consumed at the point-to-point level plays a crucial role in extending the energy life of the network, the ultimate goal of a networked environment is to allow communications to take place for applications with real quality-of-service constraints,

and achieving such a goal inevitably requires energy. The transport layer sits in between the applications that wish to exchange data, and the rest of the networking stack that strives to save energy and thus the transport layer is in the unique position to balance these needs.

In the next section we provide a detailed overview of existing literature in the area of transport protocols with respect to their suitability in wireless networks and as of their energy-efficient. We also describe how we build upon the prior work to design JTP.

2.2 Related Work on Transport Protocols

In this thesis, we propose a new energy efficient transport protocol. Although designing a protocol for Mobile Ad Hoc networks, especially while trying to minimize energy consumption is hard the area of transport protocols is a well studied area that has been the focus of research since the beginning of the packet-switching networks. In 1974 Cerf and Kahn wrote a paper [CK74] that laid the foundations for what was later to become TCP. This gives us a multitude of previous work that we can build upon and expand to face the challenges of the JAVeLEN system.

In this section we provide a comprehensive overview of transport protocol literature. In order to narrow the scope of our survey we will be focusing mainly on the energy conserving perspective of previous approaches. Our goal is to explain why current approaches can not be directly used in the Javelen system, and how our work builds on top of existing work to achieve a highly efficient transport mechanism for application data.

2.2.1 TCP, TCP-variants and other Wireline Protocols

The most commonly used protocol in computer networks to date is the Transmission Control Protocol (TCP), which was based on ideas proposed in [CK74] and is described in detail in RFC 793 [oSC81]. TCP was designed for wired networks with stable, reliable links and static topologies, and thus it is not intended to operate in the challenging environment of MANETs.

Extensive studies [FML02, AS04, HV02, BPSK97] have demonstrated the inadequacy of TCP, to efficiently operate in wireless environments. There are multiple reasons for TCP's poor performance:

- The **self-clocking mechanism** of TCP relies on the assumption that packet losses are mainly due to congestion losses, and backs off when a lost packet is detected. In wireless networks, losses due to transmission errors are more common, causing TCP to unnecessarily back off yielding low throughput.

- TCP is a **window-based** protocol which relies on the reception of acknowledgment(ACK) packets in order to transmit new data which many times lead to a burst of data packets being transmitted; imagine the case where an ACK packet verifies the reception of a large range of data, TCP would slide its transmission window significantly blasting the network with multiple packets. In a multi-hop wireless network where nodes on successive hops compete for resources this leads to contention between neighboring nodes and performance degradation.

- TCP's **dependence on ACKs** requires a steady stream of ACK packets. Both in the slow start and in the AIMD (Additive Increase Multiplicative Decrease) phases of TCP [oSC81], TCP adjusts its window based on the ACK packets that are received. In wireless networks, where the communication medium is shared the ACK stream competes for resources with the data stream yielding poor throughput.

- TCP's **timeout mechanism** for detecting severe packet loss conditions assumes that TCP operates in an environment where the Round Trip Time (RTT) is fairly stable. However in wireless networks, especially in the presence of mobility, the RTT can vary significantly, causing TCP to timeout unnecessarily yielding poor performance.

- TCP's **acknowledgment mechanism** leads to retransmission of already received data. TCP uses cumulative acknowledgments, indicating to the sender the last successfully received consecutive byte. When a retransmission is invoked the sender would

retransmit all subsequent bytes whether they were received or not.

Over the years multiple TCP variants have been proposed in order to improve the performance of TCP over wired networks. Delayed ACKs [Bar89] were proposed to reduce the stream of ACK packets. The receiver instead of sending an ACK packet for every received packet, it sends one every d packets where d is configurable. TCP-SACK [MMFR96] was proposed to address the issue of retransmitting already received packets. In TCP-SACK the ACK packet not only includes the cumulative ACK like normal TCP but it is extended to also include a list of subsequent byte ranges that have been received even if there are missing bytes in between. Other variants of TCP, like TCP New Reno [HFGN12] and TCP Vegas [BOP94] are modifications that attempt to improve the congestion detection mechanism and adjust the transmission window value accordingly. Many of these suggestions have a significant impact on TCP performance and have been adopted over the years and are now widely deployed. Although, these modifications provide a source of inspiration for designing JTP (for example the selective negative acknowledgment of JTP is very similar to the TCP-SACK mechanism), they can not be directly applied to JAVeLEN since they assume a wired infrastructure and thus do not address multiple of the wireless challenges.

In addition to improving TCP, researchers have also studied the design on new transport protocols for wired networks to overcome some of the problems of TCP. NETBLT [CLZ87] is a transport protocol specifically designed to achieve high throughput for bulk data transmission applications. Although NETBLT can not be directly used in mobile ad-hoc networks since it is optimized for an environment with very different characteristics, it provided a lot of useful insight into the design of JTP. NETBLT separates the functionality of flow control from error control and makes the receiver responsible for controlling the connection; ideas that are also adopted by JTP. The need to separate flow/congestion control from error control is also argued by the authors of DCCP [KHF06], which is a protocol that provides congestion control without reliability.

As researchers identified various TCP shortcomings, they proposed various transport protocols that are more specialized for specific services. VMTP [Che88] was proposed

to address the needs of transaction based communication, SCTP [ea00] was proposed to address signaling needs in PSTNs(Public Switched Telephone Networks), RDP [VHS84] was proposed to optimize remote debugging and application loading services. A common observation of all the above protocols is that it is more beneficial to maintain the application framing of the data while transferring it through the network instead of transforming it into a stream of bytes like TCP. This is better summarized in in a paper by David Clark [CT90b] about architectural considerations in protocol design and it is a principle also embraced by JTP. Although all these protocols improve the performance of TCP they are still designed for specific services over wireline protocols and can not be adopted as is in a MANET.

The authors of the Xpress Transport Protocol (XTP) [Wea92] build on top of the above mentioned works and try to provide a more tunable transport protocol that can be tailored to the communication at hand. XTP separates mechanism from policy and the transmission rate from flow control. The JTP follows a similar design paradigm but chooses mechanisms and policies that are optimized for wireless communication.

The problem with the lack of flexibility of current transport protocols is also discussed in the paper that introduces the Structured Stream Transport (SST) [For07]. The author argues that applications today have the choice between using UDP or TCP as their transport protocol which are fundamentally different and there is no protocol deployed that can provide applications with the flexibility of using both or something in the midst of the two. In JTP we allow for such flexibility with the use of the Application Transfer module presented in Chapter 4.

The eXplicit Control Protocol (XCP) [KHR02] is a transport protocol that was proposed for high-bandwidth optical links and more high-delay satellite links. XCP uses explicit feedback from the routers; an idea also deployed in JTP, in order to set the transmission rate, which along with other controllers running on each router, it achieves better bottleneck link utilization while maintaining very small queues. XCP needs to have good RTT estimates in order to operate efficiently which is a problem in mobile wireless networks.

2.2.2 TCP Improvements for Wireless Networks

Given that TCP is the de-facto protocol for network communications, there is a substantial amount of literature that focuses on modifications to TCP in order to improve its performance in wireless communication environments. We have classified prior work in two categories based on where in the network the new mechanism is placed: *proxy-based* for approaches that place extra functionality within the network and *end-to-end* for approaches that modify the functionality of the source and/or the destination.

Proxy-based Approaches

In wireless networks where the hosts connect to the rest of the network through a base station, most of the performance degradation happens over the last, wireless hop [CI95]. There is a significant amount of work that focuses on enhancing the functionality of the base station in order to improve the overall performance, mainly by focusing on hiding the wireless losses from the TCP sender, that would otherwise misinterpret them as congestion-induced losses.

I-TCP [BB95] and Mobile-TCP [HA97] attempt to hide the wireless losses by splitting the connection in two parts, at the base station. The base station plays the role of the TCP destination for the TCP source acknowledging successfully received packets and ensuring that all the data is received. The base station also plays the role of the TCP source for the TCP destination, by retransmitting lost packets and ensuring that data gets all the way to the destination. In this way the TCP connection is optimized for each part of the connection and the wireless losses do not affect the end-to-end performance. This approach breaks the end-to-end argument since a network device is instrumental in the successful completion of the transfer (e.g. if the base-station fails in the middle of a transfer, there is no way to recover even if a new path is found). MANETs have no infrastructure and there are no guarantees about the behavior of the hosts that are acting as routers between the end hosts, making it hard to adapt the above approaches in an ad-hoc system.

SNOOP [BSAK95] and WTCP [RM98] are good examples of approaches that attempt

to alleviate the effects of wireless losses by deploying caches at the base station. If a packet that has been received by the base station is later lost, the base station can retransmit it on behalf of the TCP source saving network resources and reducing the recovery latency. These caches are only best effort, i.e. the final responsibility for reliability still lies with the TCP source. If for example a packet is lost but the base station does not have a copy the TCP source is responsible for retransmitting the lost segment. Although these approaches can not be directly applied to MANETs due to lack of established infrastructure, one can imagine modifying these schemes to operate without a base station. For example, JTP inspired by SNOOP, deploys best-effort caches in all the nodes in an attempt to recover losses as close to the destination as possible. However, the interaction between these schemes and those deployed in the link-layer to improve link reliability (e.g. ARQ [CC84]) are not straightforward. Ludwig has shown that, if not designed carefully, end-to-end and in-network retransmissions, used together, can cause worse performance than either alone [Lud00b].

In the context of proxy-based schemes, the tradeoff between throughput performance and energy costs (due to transmission power and error control) are analyzed in [BMAA04]. The authors examine the effects of power transmission management, redundancy and retransmissions at the link layer on TCP performance and conclude that improvements are always associated with a cost and improvements should not be made in isolation but in a coordinated fashion.

End-to-end Approaches

In these approaches the sender is usually trying to identify the type of loss; whether it is due to congestion or due to the condition of the channel. Some of the approaches use *explicit-feedback*, where the sender is informed about the state of the network by the routers. Approaches in this category include TCP-F [CRVP98], ELFN [HV02]), ECN [RFB99], ELN [BK98], and ATCP [LS01].

Another set of approaches employ *implicit-feedback*. The sender attempts to distinguish

the reason for the loss, by measuring end-to-end delivery statistics. In WTCP [SVSB99] for example, the authors propose to use inter-packet delays for congestion control. On the other hand ADTCP [GML02] uses multiple metrics (including packet loss rate, short-term throughput and others) to identify the current state of the connection and the network. Biaz and Vaidya propose a scheme in [BV99] that is deployed at the receiver in order to discriminate congestion losses from corruption ones. This scheme works only if the last hop to the receiver is wireless and if the wireless link is the bottleneck, which are reasonable assumptions for wireless networks but not for MANETs.

Although these approaches strive to improve TCP's performance, even perfect knowledge of the reason for packet loss (*e.g.* congestion-induced vs. transmission error) at the sender, often, does not improve throughput performance [KSE+04, BM02]. Moreover, these schemes suffer from the slow adaptation of TCP's AIMD mechanism [Jac88] to the fast changing conditions of wireless links. TCP-Westwood [CGM+01] addresses this problem by augmenting AIMD with an estimate of the available bandwidth measured, based on the ACK reception rate. Other approaches try to alleviate the effects of bursty TCP traffic by clamping the congestion window [AHM03] or by pacing TCP packets [ASA00]. Although these approaches significantly improve TCP performance they still rely on packet loss to identify congestion, and are not appropriate for an energy-conscious environment.

2.2.3 New Transport Protocols for Wireless Networks

The self-organizing nature of MANETs makes them ideal for situations where there is no or limited infrastructure. Example applications include tactical MANETs, disaster relief scenarios, and sensor systems. In all these scenarios the ad-hoc network can be isolated and its main goal is to maintain connectivity between its members rather than ensure high performance with an external network. This observation has led many researchers to explore new transport protocols that are not based on TCP, since backward compatibility with widely deployed systems is not necessarily a concern. In this section review such clean slate approaches. We have classified these protocols into the following categories: *rate-based*

flow control for protocols whose transmission is rate-based and not window-based like TCP, *application-specific protocols* that are designed to support a specific type of application, *energy-aware schemes* and protocols designed specifically for *sensor networks*. We also have a last category for *other* protocols that do not fit in the above categories.

Rate-based Protocols

The self-clocking mechanism of TCP causes data and ACK packets to be sent in bursts, leading to channel contention and packet drops even before the link is congested. To ameliorate this problem, rate-based protocols have been proposed, whereby the available along the path to the destination could be explicitly collected and fed back to the sender.

ATP [SAHS03] is a reliable transport protocol specifically designed for MANETs. ATP uses rate-based transmissions and just like NETBLT it separates the congestion from the error control. In order to properly set the rate, ATP relies on feedback from the intermediate nodes and more specifically on the cumulative (queuing and transmission) delay that a packet experiences at each hop.

EXACT [CNV04] is another scheme where a flow's allowed rate is explicitly conveyed by intermediate nodes. EXACT requires per-flow state at the routers and thus it is not suitable for systems with a large number of flows.

RBCC [ZCF05] uses the channel busyness ratio as a sign of the network utilization and congestion status. RBCC is basically a sublayer that is placed under TCP in order to control the rate at which packets are released into the network.

All the above techniques use a mechanism akin to DPS [SZ99] where the feedback from all the intermediate routers is accumulated in a specific header field in protocol packets. JTP employs a similar mechanism.

Although these rate-based solutions outperform existing TCP variants and avoid bursty traffic, they still use frequent *constant-rate* feedback which competes with data flows for resources.

Application-specific Protocols

Just like in wired networks, there have been transport protocol proposals in wireless networks that are cognizant of *certain* QoS requirements of the application.

A good example is MRTP [MBNP06] which is the wireless enhancement of the RTP [SCFJ96] protocol for real-time applications. MRTP leverages the existence of multiple paths between the sender and the receiver in order to provide improved media quality in wireless networks.

JTP draws inspiration from these protocols and as described in Chapter 3, JTP provides enough flexibility to the application so that it can better support different application requirements.

Energy-conscious Protocols

In all aforementioned related research, energy consumption has not been considered as a constraint in the transport protocol design. An important step in optimizing the energy consumption in a system is by turning off components when they are not used.

Researchers have looked into ways to turn off the network interface on hosts when it is not used, without compromising the application's requirements. Most power management schemes operate at the MAC layer by turning off the radio when there are no data to be transmitted. However, the MAC layer is agnostic to the data in the packet, and thus there have been studies about enabling the applications to influence the radio schedule. The authors in [BN00] and in [BRBR03a] propose modifications to the transport layer protocol so that it manages the communication device based on run-time parameters of the transport protocol. Kravets et al. [KK00] propose a new transport layer protocol that turns off the radio for short periods of time while informing the base station so that any packets will be queued for the period of time that the communication device is off.

Steinbach, in his master thesis [Ste02], proposes a new protocol that sits between the transport and the MAC layers. This protocol uses reinforcement learning to predict future traffic patterns by monitoring the existing traffic. This approach does not require any modifications to the applications but still performs better than MAC layer approaches.

The authors in [CV02] propose not only to monitor the application data but to also shape the data transmission so that it allows the communication device to transition to the sleep state.

The Wave and Wait Protocol (WWP) [TBV99] was proposed as an energy-saving transport protocol for data transmission that is not delay sensitive. The idea behind WWP is that the sender will first probe the condition of the network and it will only transmit if the data has high probability of getting to the receiver without being dropped in the network. On the other hand GreenCall [NG10] is a transport protocol that is aimed to support Voice over IP (VoIP) calls over wireless networks while conserving energy. GreenCall, just like JTP, exploits the fact that not all data has to be received in order for the VoIP call to be successful. PGTP [ASM+11] is an energy-aware transport protocol specifically designed for multi-player mobile games. PGTP takes into account the current state of the game in order to decide whether the data packets can be buffered and sent at a later time.

Although all the above techniques are hard to apply in multi-hop wireless networks, where each node is both a router and an end-node, they nevertheless indicate that application-aware protocols can achieve higher energy savings.

We demonstrate in this thesis, that even if network nodes are parsimonious in their use of energy (*e.g.* nodes turned off when there is no data to transmit or receive), an energy-aware transport protocol, such as JTP, can achieve greater energy gains by turning on the radios only when it is absolutely necessary. To this end, JTP minimizes control traffic and avoids data transmissions that are unnecessary for meeting given delivery requirements of applications.

Sensor Protocols

Wireless sensor networks are a specialized form of ad-hoc networks where battery operated devices are placed to perform specific monitoring tasks and can communicate wirelessly with each other and with the collection station. While sensor network deployments are often highly customized such that they are not suitable for general purpose networking,

their solutions may still provide insights into designing protocols for MANETs since they face many of the same challenges.

More specifically sensors are battery operated and they are expected to operate long after they are deployed so it is important to be energy conserving in order to maximize the lifetime of the network. Energy-aware transport protocols have been proposed in the realm of sensor networks, such as PSFQ [WCK02] and RMST [SH03]. In these approaches, emphasis has been given to the local recovery of lost packets from local caches, whereby each node must monitor every flow for losses. Given the goal of one-to-many reliable delivery in such a sensor network realm (*e.g.* to program the sensors), issues that arise in multi-hop wireless networks regarding the fair allocation of resources among flows and the reduction of in-network overhead have not been considered.

PCCP [WLS+07] and IFRC [RGGP06] are congestion control protocols for WSNs. Both of these approaches deploy a hop-by-hop congestion control protocol that is based on local measurements tries to alleviate congestion in a node's neighborhood.

A common theme in transport protocols that are designed for WSNs with the goal to conserve energy as well as provide reliability is that they place certain functionality within the network. JTP follows a similar design philosophy and places the appropriate functionality within the network in order to achieve more efficient use of network resources.

Other Protocols

Recently there has been an effort to create practical implementations of the theoretic result of [TE92]. Diffq [WJHR09] adapts the results of [TE92] to design a backpressure congestion control protocol and implement it in off-the-shelf radios. Diffq is located between the transport and the network layer and the authors have implemented Diffq-compatible TCP and UDP versions so that current applications can run on top of a system running Diffq. XPRESS [LSLLG11], on the other hand is a new complete MANET architecture that tries to implement [TE92], using TDMA MAC protocol and by tightly integrating the transport and network layers. The original theoretical work did not account for energy

consumption so although these approaches do achieve great performance improvements in terms of throughput, they can not be directly applied in a system striving to save energy.

2.3 Conclusion

JTP was mainly motivated, and finally implemented within the JAVeLEN system, that is a highly energy conserving platform for MANETs. In this chapter we have provided an overview of the JAVeLEN system and the different protocols deployed. JAVeLEN is designed to save energy in all levels of the system and it provides multiple orders of magnitude in energy savings compared to state of the art systems. This highly energy conserving platform poses a challenge to network protocol designers to try and amplify the energy savings. In this chapter we have also reviewed the literature in transport protocol design and presented the insights that helped us design JTP.

JTP inspired by XTP [Wea92] separates policy from mechanism. By this separation JTP's design, see Chapter 3 for details, is generic and can be applied in any network that aims to reduce the energy consumption. JTP uses a DPS-like [SZ99] mechanism to record and report network statistics to the end points of a connection. Furthermore, JTP expands the idea of in-network caching deploying a pipeline of caches between the sender and the receiver striving to save energy by recovering lost packets as close to the destination as possible, see Chapter 5. JTP builds upon all these ideas and combines them in a unique way to save energy in a MANET environment.

Chapter 3

Design and Architecture

3.1 JTP Design

In a network architecture some layers lend themselves better for direct improvements toward a given performance metric than others. In a network aiming to improve its energy efficiency the physical layer can be focused on aggressively reducing the energy-per-bit transmit/receive requirements. Similarly, the routing layer can aim at establishing routes that minimize the overall energy consumption. At the transport layer however, this type of optimization metric may come in conflict with the requirements of applications (reliability, low delay, high throughput, etc.) The transport protocol is responsible for addressing the challenge of optimizing the energy consumption while still satisfying the application's requirements. This challenge is even more tricky if we consider that different applications have conflicting requirements. For example, a bulk data transfer requires high reliability but delay is not a concern, while a VoIP application requires low delay and reliability is less important.

The total energy consumed by a transfer is generally affected by the route chosen, by link-layer decisions, or by other competing network traffic[1]. In JAVeLEN each layer is optimized to conserve energy which contributes significantly to reducing the total energy

[1]Competing traffic might lead to collisions during transmission and packet drops due to network congestion. Lost packets during a transfer might lead to retransmissions, which affects the total energy consumed.

consumed during a transfer. Most importantly, JAVeLEN implements a distributed, fair-access, collision-free MAC protocol among nodes [RCP+04, RKM+08], which effectively minimizes the adverse interactions between competing traffic. Thus one can aim at reducing the network-wide energy usage by reducing the total transmissions that are necessary for each transfer – an elegant bit of simplicity. However, the problem is still difficult because very little is known about minimizing transmissions across multiple nodes. The one key piece of work in the area, Ludwig's work on the interaction end-to-end and hop-by-hop retransmissions [Lud00b], gives a depressing result: namely in a world where one needs both end-to-end and hop-by-hop retransmissions, it is very easy to have the two transmission mechanisms interact to cause more, rather than less, total transmissions. Without a strong base of prior work to build upon we developed a set of complementary mechanisms, each of which contributed to reducing the total number of transmissions, resulting in a protocol that works well, as the experiments show.

Following is a list of design goals and the corresponding choices in developing JTP, based on the above observations.

- *Enable the support of diverse applications.* Transport protocols usually offer a particular reliability/QoS model and the application has to choose which protocol is more appropriate (e.g. UDP, TCP, ITP [RBS00], RTP [SCFJ96]). JAVeLEN is envisioned to support a variety of applications, from bulk data transfers, to VoIP and video streaming. One approach would be to design and implement multiple different protocols each optimized for a specific type of application. However, we discovered that when striving to conserve energy many design decisions are common independent of the application that is being supported, and many of the transport protocol functionalities (connection establishment, data integrity, etc) are also shared between different protocols. These observations lead us to the decision that it is more efficient to design on transport protocol that can be tuned appropriately in order to support different applications. JTP is able to optimize the network energy consumption for any type of application to achieve the goal of JAVeLEN by design. JTP cleanly separates

the **application-dependent** part of the transfer control from the rest of JTP. This part of JTP is responsible for communicating with the application, understanding the QoS requirements (loss tolerance and delay) and translating them to specific network choices. This is described in greater detail in Chapter 4.

- *Minimize end-to-end retransmissions.* Ludwig's work showed that we need to strike a balance between end-to-end (source) and local retransmissions. End-to-end retransmissions effectively waste all the energy already expended on getting the packet at least part way to the destination by the initial transmission. So while occasional retransmissions from the source are required (e.g. due to intermediate node failure or topology changes), we seek to do everything we can to retransmit a lost packet from the farthest downstream node along the path which had received the packet successfully. To achieve this, we deploy **opportunistic caches** on all nodes (Chapter 5) that store received data packets.

- *Minimize link-layer retransmissions.* It is common in wireless networks to deploy ARQ [CC84] in an effort to improve the perceived link quality. Although such a feature is helpful since it's better to correct an error exactly when it happens, the number of retransmissions is usually static and is set irrespective of the applications requirements. As pointed out by Ludwig, this can have adverse interference with end-to-end efforts of recovery, as well as affecting the total packet delay. JTP employs a **dynamic ARQ** scheme that adjusts the number of link-layer retransmissions based on the tolerance of the application to losses (Chapter 5).

- *Minimize acknowledgments.* Acknowledgments are, often, pure overhead—they carry no application data, yet they consume roughly as much energy as a data transmission; the fixed overhead cost of sending or receiving a packet is relatively high compared to the incremental cost related to its the size [FN01, CSW03]. Consistent with reliability and other goals, we endeavor to reduce acknowledgment traffic. JTP employs a **destination-centric** design where the destination, and not the source, controls

the transfer (transfer rate, packet retransmissions, etc.) Moving the control to the destination enables JTP to deploy a **variable feedback** scheme that aims at keeping feedback as low as the stability and reliability of the network permits (Chapter 6).

- *Avoid congestion loss in the nodes.* A classic TCP-like congestion control induces packet drops in order to enable detection of congestion. The energy expended by packets that are discarded simply to signal congestion is wasted. In a world where energy is the key metric congestion control must consume less energy. JTP is a **rate-based** protocol that aims at avoiding congestion rather than controlling it. The transfer rate is set based on the minimum *available* capacity on the path, striving to avoid any congestion losses (Section 3.2.2).

3.2 Architecture

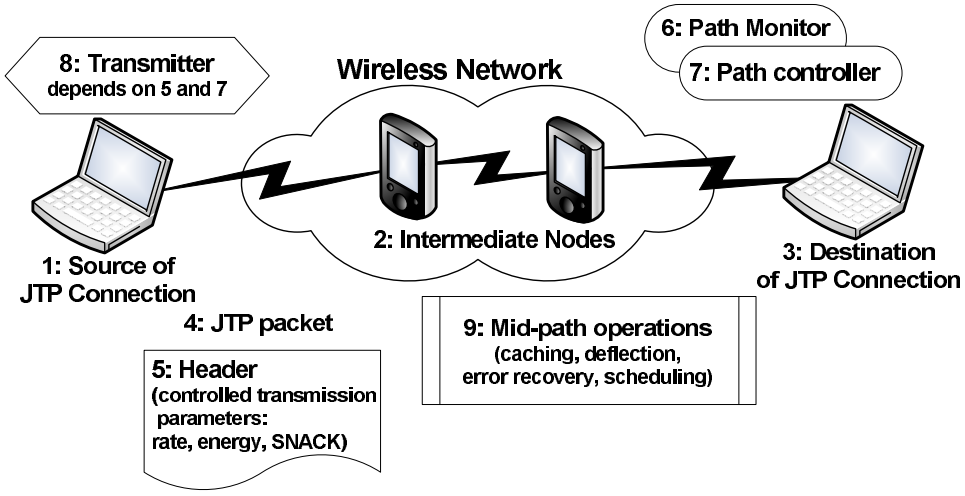


Figure 3.1: Elements of JTP

Figure 3.1 shows a high level architecture of JTP along with all the elements and mechanisms and how they are placed in the network. JTP uses *rate-based* transmissions controlled by the destination. Intermediate wireless nodes *report on their condition* in *packet headers*. *Path Monitor* and *Path Controller* at the destination collect path-performance data and adjust the data sending rate to avoid congestion. When the parameters of the connection

have significantly changed, the destination notifies the sender of the new transmission parameters and also about data packets that need to be retransmitted. Intermediate nodes *cache packets*, *examine end-to-end acknowledgments* and *retransmit packets* on a per hop basis as needed. If an acknowledgment indicates a packet was lost farther along the path, the node would *retransmit* its cached copy, thereby avoiding an end-to-end retransmission (a la the work of Balakrishnan *et al.* [BSAK95]).

Figure 3.1 provides a horizontal view of JTP, showing where the different mechanisms lie in a JTP connection. Figure 3.2 depicts a vertical JTP view, presenting how JTP is organized within each node: what are the modules, where they are located, and how they interact with each other.

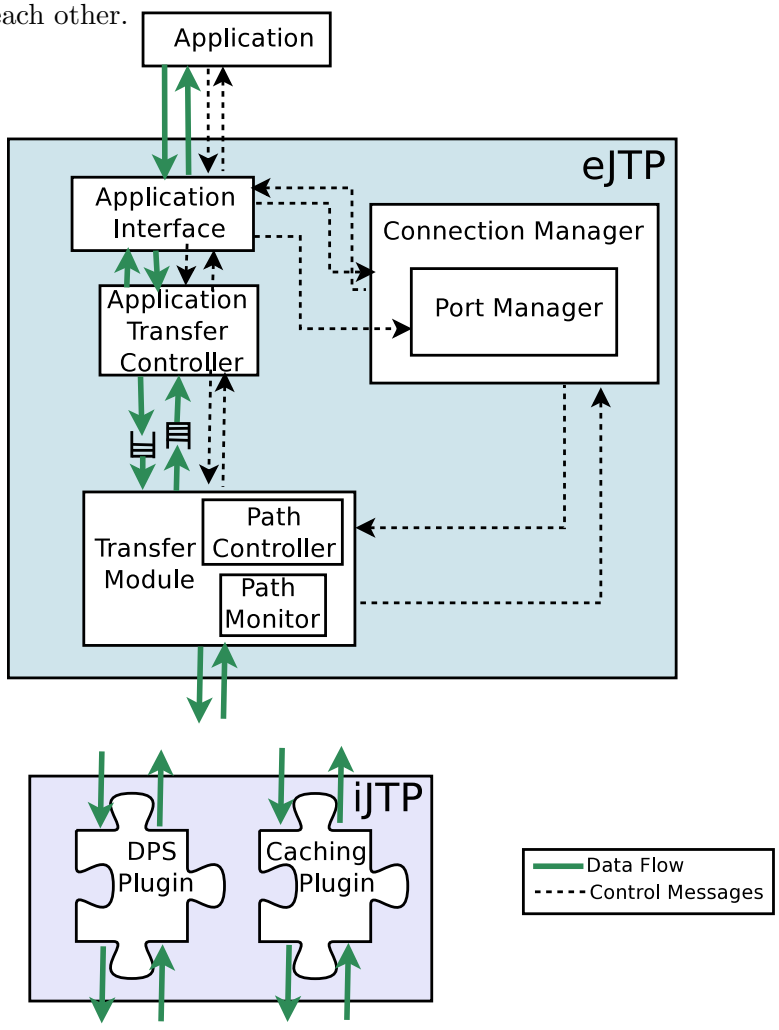


Figure 3.2: Modules of JTP

As mentioned earlier in the introduction (Chapter 1), although JTP is an *end-to-end* transport protocol, it also performs hop-by-hop operations to improve its performance. In order to keep the architecture clean and to allow different deployments, JTP is divided into two main components: the *end-to-end JTP* (eJTP) and the *hop-by-hop (or, intermediate) JTP* (iJTP). The functions of each component are kept separate and are designed to work independently. This separation allows for efficient JTP implementations that minimize processing delays inside the network. iJTP functions are invoked on every node in the path and are executed in the fast path of forwarding packets and thus should be kept simple such that they do not increase the processing delay of packets. In a slotted MAC like the one used in JAVeLEN this is very important because if the processing delay of a packet is not bounded it can lead to nodes 'blowing' slots, i.e. not completing the transmission of a packet within the duration of one slot. eJTP on the other hand is not on the fast path; it is executed before a packet enters the network at the source, or after it is received at the destination. eJTP can be more complex without affecting forwarding delays.

3.2.1 eJTP: End-to-end JTP

The mechanisms that are part of eJTP operate end-to-end, and are invoked only at the end hosts (the source or the destination) of a JTP connection. eJTP, as depicted in Figure 3.2, is comprised of various modules that are described below in detail.

The Application Interface

This module is responsible for interacting with the application. It provides a set of functions that the applications invoke in order to send and receive data through the network using JTP. The Application Programming Interface (API) follows closely the Linux socket API [SFR03] to enable easy porting of JTP as a Linux kernel module. The API is described in great detail in Appendix A.

As shown in Figure 3.2, the Application Interface (AppIntf) module communicates with the *Port Manager module* in order to get/assign ports from/to new connections, with the *ap-*

plication transfer controller to send/receive data and with the *Connection Manager module* to create/use/destroy JTP connections.

The Application Transfer Controller (ATC)

This module enables JTP to support a variety of applications with different requirements. This is the module that is responsible for transforming data from the application to JTP data packets. As pointed out in [CT90a] not all applications have the same requirements in terms of how and when data should be delivered to them, as long as they receive them in a form that they can parse. In JTP the communication with the applications is in terms of Application Data Units (ADUs). This Application Level Framing [CT90a] (ALF)-inspired structure supports a variety of application-dependent policies which determine how many ADUs get packed in one JTP packet, how to deliver successfully received ADUs to the application (e.g. unlike TCP, out-of-order delivery could be allowed), which ADUs are more important than others, whether the communication is unidirectional or bidirectional, and other requirements such as reliability and timing constraints on the delivery of ADUs so they are useful to applications.

Along with the application data in packets ATC also provides the necessary information to the network about how the packets should be handled based on the application's requirements. ATC is also responsible for deciding which packets should be retransmitted in the face of losses.

In order for JTP to be able to support multiple different applications, it supports running multiple instances of different ATCs. Each application can choose which ATC controller to use through the application interface. If the application does not actively choose one, a default one is assigned. More details about this module are presented in Chapter 4.

This module stands between the applications and the actual transfer functions of JTP. The data flow between ATC and the transfer module is asynchronous and buffers are used to ensure that packets are not lost in between the two modules, as shown in Figure 3.2.

Connection Manager

This is the module that maintains JTP connections. It is responsible for monitoring the state of each connection and acting accordingly.

As mentioned earlier in Section 3.1, JTP strives to minimize the control overhead. Although JTP connections have explicit **Connection Establishment** and **Connection Termination** phases JTP allows applications to piggy-back data on any control packet. Imagine that a sensor application is running and once a minute it reports the temperature. In this case the sensor can establish a connection, send the data, and terminate the connection with only two packets; the source will initiate, terminate the connection and send the data in one packet and the destination just has to acknowledge the termination of the connection with another packet. For more details on packet types see Section 3.2.3.

The reduction of control traffic for establishing and tearing down a connection brings up the issue of the integrity of the connection, i.e. reliable connection management. One of the main reasons for the three-way handshake of TCP is to ensure that both the sender and the receiver are synchronized and it is guaranteed that packets from old connections can not interfere with the newly established one. Watson explored the connection management problem with fewer than the five packet required by TCP in his Delta-t [Wat89] work. Basically Delta-t uses timers both at the sender and at the receiver to ensure proper management of each connection. JTP takes a very similar approach, where it ensures integrity of a connection by employing both randomization of initial packet sequence numbers as well as appropriate timers to ensure that the connections are properly established and terminated.

The Port Manager, which is a submodule of the Connection Manager, is responsible for keeping track of assigned JTP ports. In Linux systems the kernel already provides this feature as a service and thus the Port Manager can be deactivated. However, JTP is envisioned to be deployed in embedded platforms where the operating system would only provide basic functionality, and thus needs to be able to work independently. The Port Manager only interacts with the Application Interface module in order to assign ports to

connections.

An application can request a specific port to bind to – which is usually used for server applications – or let the Port Manager assign one. The problem of assigning ephemeral ports has been studied before [ML11, All09] and JTP employs common practices [ML11] of randomizing the assigned ports in order to avoid collisions and to be secure against "blind" attacks [Shi07].

Transfer Module

The transfer module is the module that handles incoming packets, processes them and demultiplexes them to the appropriate JTP connection. The module is also responsible for sending packets out on behalf of the applications based on the transfer parameters. These parameters are monitored and adjusted by the *Path monitor and Path controller* that are both part of the transfer module; more details on how these two components work are presented in Chapter 6. The transfer module is the main module that interacts with the rest of the JAVeLEN system.

3.2.2 iJTP: Hop-by-hop JTP

iJTP includes all the JTP operations that are performed by the mid-path nodes. Ad hoc networks differ significantly from other networks since the devices that are performing the routing and forwarding of packets in the network are themselves hosts and thus already running the full networking stack. In traditional networks (e.g. the Internet), adding functionality within the network usually entails redesign and enhancements in routers and switches. On the other hand, in ad hoc networks, all the functionality is already present in mid-path devices, enabling protocol designers to more easily add functionality in the network as long as it maintains efficient processing of packets.

All iJTP operations are soft-state [Cla88] and do not require any per-flow state to be kept, which scales well with the number of flows. iJTP has two main modules:

- **caching module** performs all the *cache-related* operations, storing and retransmitting

data packets based on received feedback (ACK) packets.

- **DPS module** inspects/alters JTP packets based on information stored in the packet or any network information available; there is no per-flow or hard state kept.

Besides the fact that iJTP must process all JTP packets that pass through a node, the soft-state operations require the crafting of cross-layer interactions with the MAC layer (e.g. iJTP needs to know the available capacity to each neighbor). In order not to compromise the performance of a node by redundant copying, context switching, and message passing between the transport and the link layers, we implemented iJTP as separate, loadable plugin modules of JAVeLEN MAC, a mechanism explained in Section 2.1.2. Although within the JAVeLEN system iJTP resides in the MAC, iJTP operations can alternatively be performed at the transport layer using an overlay architecture.

The JAVeLEN MAC plugins are invoked right **before** and **after** a packet is transmitted and right **after** a packet is received. A high level pseudocode that describes the combined operations of the two plugins is presented with Algorithm 1 and Algorithm 2. These algorithms only present an abstract view of the functionality and are provided to give to the reader a perspective of the operations performed. In summary before a packet is transmitted, some of the fields are updated based on the DPS plugin and the effort this node will put in transmitting this packet is also decided while after a packet is received the caching plugin decides whether to cache a copy of this packet and whether some of the cached packets need to be retransmitted. For a more accurate representation of the implementation the reader is referred to the detailed flowcharts for each plugin that follows the description.

Caching Plugin

The *Caching Plugin* manages the local cache at every node. It ensures that only valuable data packets are stored; packets that are no longer viable (i.e. no longer useful to the application) are evicted from the cache. The cache has a limited size that is configured at

Algorithm 1 $PreXmit()$

```
1: if firstDataTransmission(packet) then
2:     lossRate = getLinkLossRate(packet);
3:     setMaxDataTransmissions(packet, lossRate);
4:     updateLossTolerance(packet);
5: end if
6: rate = getAvailableRate(packet);
7: packet.rate =
8:     MIN(packet.rate, rate/AvLinkLayerAttempts);
```

Algorithm 2 $PostRcv()$

```
1: if (packet.type == DATA) then
2:     cachePacket(packet);
3: else if (packet.type == ACK) then
4:     retransmitPackets(packet.SNACK)
5:     updateACK(packet);
6: end if
```

boot time on every node. When the cache is full, packets are evicted based on the selected caching policy; see Section 5.3.3 for more details on cache replacement policies. Consistent with the design choice of distinguishing mechanisms from policies, the design of the caching supports the implementation of various caching replacement policies that can be exchanged based on the target environment. Each caching replacement policy must implement the following main functions:

- **InitCache** that initializes the cache data structure
- **DestroyCache** that destroys the cache
- **InsertPacket** that inserts a packet in the cache
- **EvictPacket** that removes a packet from the cache

In the current implementation only one cache replacement policy can be active, and it is configured at the boot time of every node. However, the design supports the dynamic switching between different policies at runtime. Chapter 5 provides more details about how the caching mechanism works.

When a data packet is received the caching module decides whether to cache it or not (Algorithm 2, line 2). On the other hand, if the caching module receives a feedback packet (ACK) it retransmits all requested data packet from its cache (Algorithm 2, line 4) and updates the ACK packet to indicate which packets were retransmitted (Algorithm 2, line 5).

Flowcharts for the Caching Plugin. As shown in Figure 2.2, each plugin must provide handles for three function to the JAVeLEN MAC; *PreXmit, XmitComplete, PostReceive*. The caching plugin does not perform any operations before a packet is transmitted and thus the PreXmit handle is Null.

XmitComplete Flowchart for the Caching Plugin. Figure 3.2.2 represents caching's XmitComplete function. All plugin functions first check to verify that this is a JTP packet. The next action the caching plugin needs to take after that transmission of a JTP packet is to check whether the packet needs to be added to the cache. If the node is the source of the packet then there is no reason for the caching plugin to cache the packet, since eJTP maintains a copy. After that the plugin performs a check that the packet has not expired yet. If it has expired then we remove the packet from the cache. If not, then we check if this was the last attempt for the packet

To make it easier to explain the functionality of each flowchart, we have assigned a number in each block. The main functionality of the XmitComplete(), Figure 3.2.2, function is to remove expired packets from the cache:

1. This is the entrance block to the function.

2. The first thing that happens is to check whether the packet that was just transmitted is a JTP data packet or not. If it is not then the function simply returns.

3. If the packet is a JTP data packet, then we check to see if the current node is the source. If it is indeed the source, then there is no reason for the caching plugin to take any action since the source has the responsibility of the correct transmission of this packet at the JTP layer.

4. If the current node is not the source, then we check if the packet has missed its deadline.

5. If the packet has missed its deadline then we remove it from the cache since the packet is not useful anymore to the application.

6. If the packet is still valid then we just return.

Figure 3.2.2 represents caching's PostRcv function. To simplify the flowchart the step of processing the SNACK has been abstracted and presented on a different flowchart, Figure 3.2.2. The main functionality of PostRcv() is to cache received data packets and to ensure that packets that are requested for retransmission by the destination are resent:

1. This is the entrance block to the function.

2. The first thing to do is to verify that this is a JTP packet.

3. If this is a JTP packet, then we check whether it is a SNACK or a DATA packet. If it is none of the two then the function simply returns.

4. If the packet is a feedback packet then we process the selective negative acknowledgment and transmit any requested packet.

5. If the packet is a JTP data packet, then we check to see if the current node is the source. If it is indeed the source, then there is no reason for the caching plugin to take any action since the source has the responsibility of the correct transmission of this packet at the JTP layer.

6. If the current node is not the source, then we check if the packet has missed its deadline.

7. If the packet has missed its deadline then we remove it from the cache since the packet is not useful anymore to the application.

8. If the packet has not missed its deadline then it is added to the cache.

9. And finally the function returns.

The ProcessSnack functionality is depicted in Figure 3.2.2:

1. This is the entrance block to the function.
2. Check the packet to see if there are any packets that are requested for retransmission.
3. If there are packets for retransmission then parse the SNACK in the received packet to get the list of packets that the destination wants to be retransmitted.
4. Extract a packet from the list.
5. Check if there was a packet in the list and whether the packet is in the cache.
6. If it is in the cache then update the packet to reflect that this node is going to retransmit this packet,
7. and then pass the packet for retransmission to the JAVeLEN MAC.
8. If there are more packets for retransmission go to back to block (4).
9. When the list is empty, return.

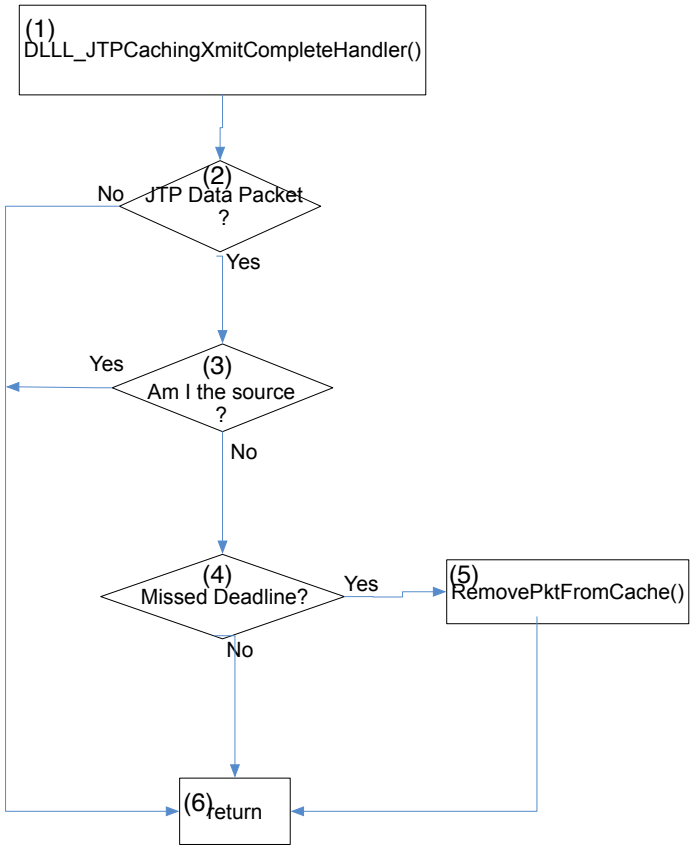


Figure 3.3: The flowchart of the operations that the caching plugin performs right after a packet has been transmitted

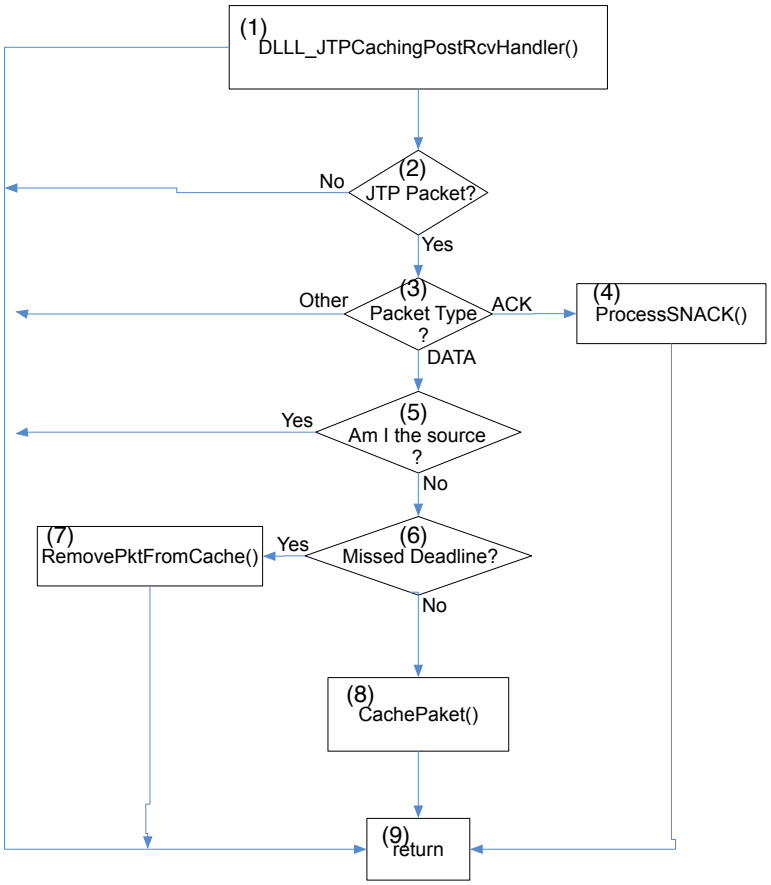


Figure 3.4: The flowchart of the operations that the caching plugin performs right after a packet has been received

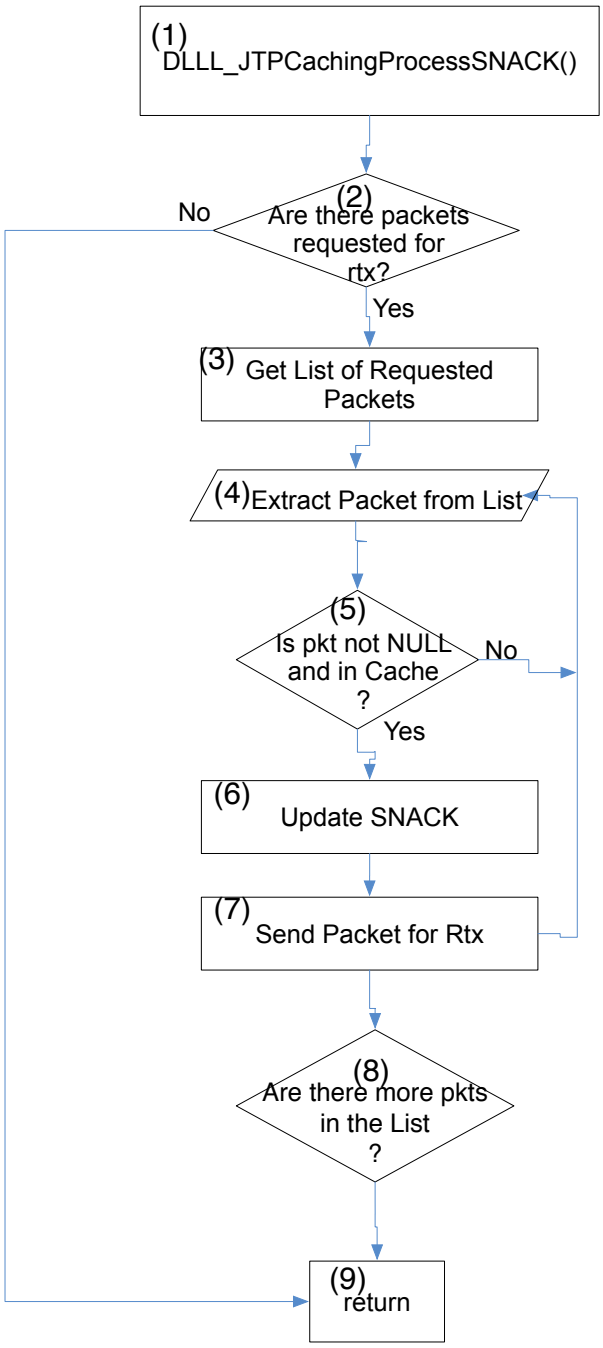


Figure 3.5: The flowchart of how the caching plugin processes a feedback packet, and particularly the Selective Negative ACKnowledgement (SNACK) fields

Dynamic Packet State (DPS) Plugin

The Dynamic Packet State plugin inspects and updates various packet fields a la the work of [SZ99] in order to record path characteristics in the packet for the receiver to analyze. Although a detailed explanation of the packet format is presented in Section 3.2.3, we briefly describe the operations that DPS performs on the packets.

- *Update the available rate field*: iJTP is responsible for acquiring from the MAC layer an estimate of the available rate to every neighbor, as well as an estimate of the average number of times a packet is retransmitted at the link layer. iJTP needs both of these data points in order to estimate the *effective* available rate to each neighbor (Algorithm 1, line 12). The average number of retransmissions is important since the available capacity that the node has to transmit new packets is affected by the number of times the new packet will have to be retransmitted. For example, if each packet were to be transmitted twice, it is straightforward to see that the effective throughput that can be achieved is half of the maximum capacity. iJTP stamps each passing packet with the lowest effective available rate (throughput) observed so far along the path.

- *Update the loss tolerance field*: Based on the loss tolerance carried in the packet's header and the link's estimated packet loss rate, iJTP sets the number of data transmission attempts on that link (Algorithm 1, line 7), as we elaborate in Chapter 5. The loss tolerance field is then updated to reflect its value for the remainder of the path.

- *Discard expired packets*: If a packet has missed the deadline then the packet is dropped and is not transmitted further. Expired packets are not cached.

Flowcharts for the DPS Plugin. As shown in Figure 2.2, each plugin must provide handles for three functions to the JAVeLEN MAC: *PreXmit, XmitComplete, PostReceive*. The DPS plugin registers all three functions represented in Figures 3.2.2, 3.2.2, 3.2.2

respectively.

Figure 3.2.2 shows the actions that the DPS plugin takes right before a packet is transmitted by the radio:

1. The first block is the entrance to the function.
2. Check if this is a JTP packet, and if it is not, exit the function.
3. If this is a JTP packet check if the packet has missed its deadline.
4. If it has missed its deadline drop the packet.
5. If the packet is still valid, check if this is the first attempt of the JAVeLEN to transmit it, and if it is not, jump to block (7).
6. If it is the first time then the DPS plugin sets the maximum number of link-layer retransmissions.
7. Then the DPS plugin updates the available Minimum Rate field of the packet.
8. Finally the function returns.

Figure 3.2.2 shows the actions that the DPS plugin takes right after a packet is transmitted by the radio:

1. The first block is the entrance to the function.
2. Check if this is a unicast packet (note that it does not have to be a JTP packet.)
3. If the packet is unicast then update the Packet Loss Rate statistics that are kept by JTP.
4. Check if this is a JTP packet, and if it is not, exit the function.
5. Check if the packet has missed its deadline.
6. If the packet has missed its deadline then drop the packet.

7. Finally the function returns.

Figure 3.2.2 shows the actions that the DPS plugin takes right after a packet is received by the radio:

1. The first block is the entrance to the function.
2. Check if this is a JTP packet, and if it is not, exit the function.
3. Check if the packet has missed its deadline.
4. If the packet has missed its deadline then drop the packet.
5. Finally the function returns.

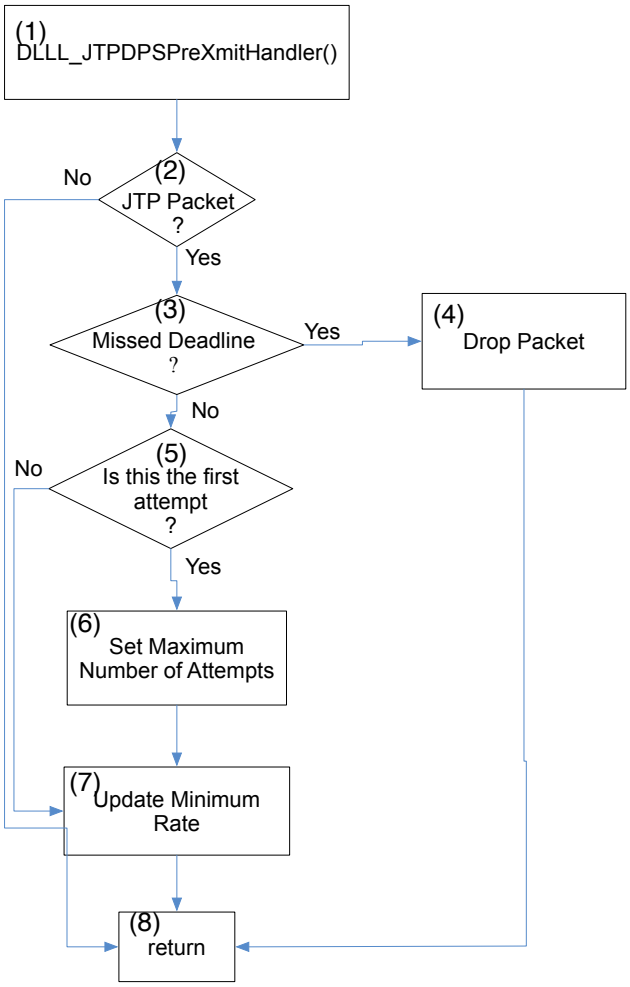


Figure 3.6: The flowchart of the operations that the DPS plugin performs right before a packet is transmitted

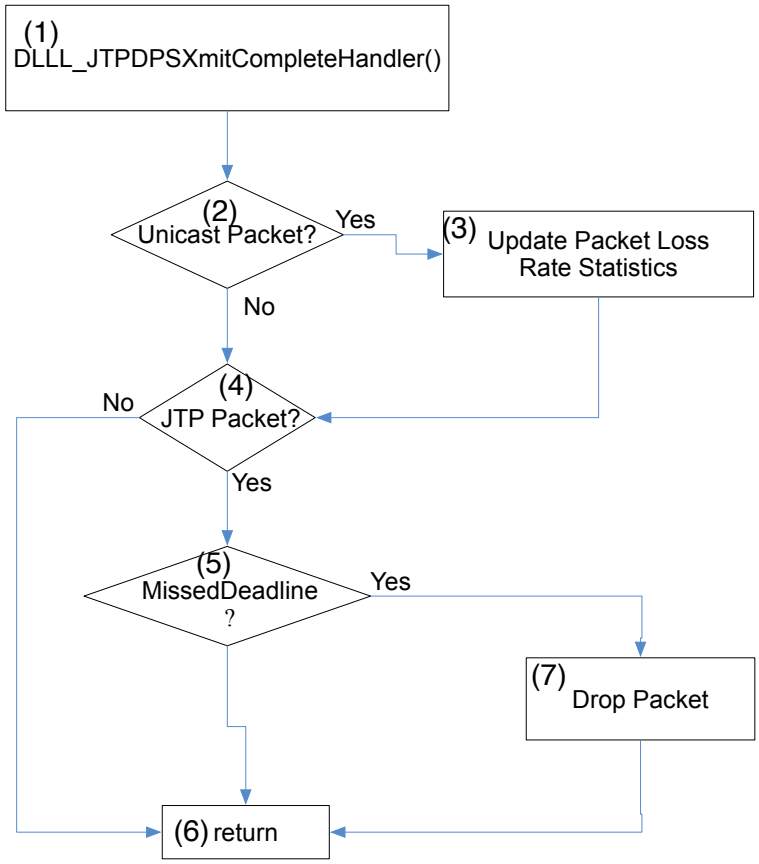


Figure 3.7: The flowchart of the operations that the DPS plugin performs right after a packet has been transmitted

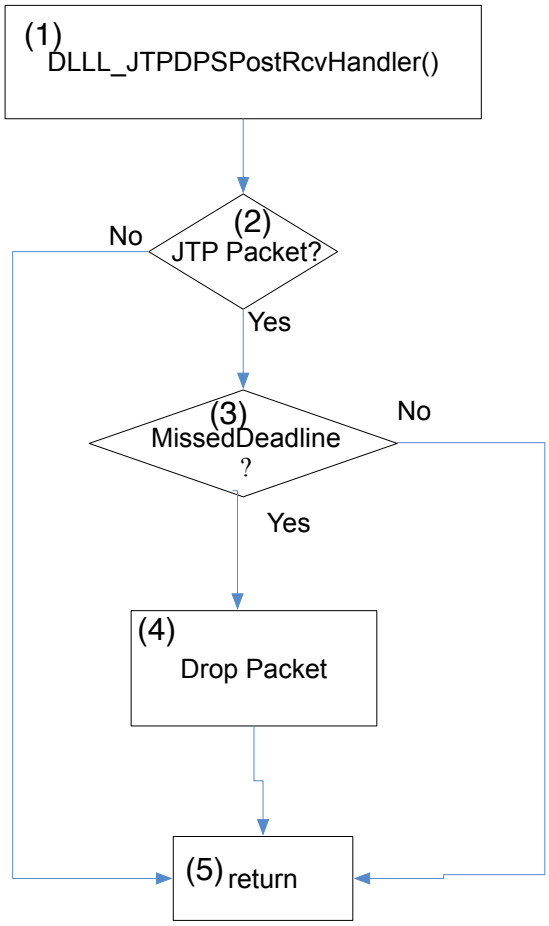


Figure 3.8: The flowchart of the operations that the DPS plugin performs right after a packet has been received

3.2.3 A Packet-based View of JTP

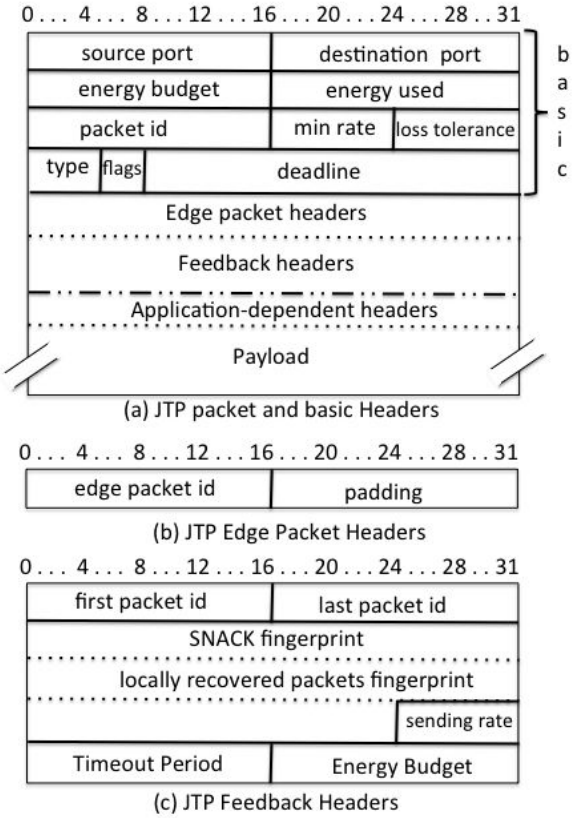


Figure 3.9: JTP packet format

In this section we present how JTP operates by describing how data flows through the network during the life of a JTP connection. Figure 3.9 shows the packet format for JTP packets. The basic JTP header, shown in Figure 3.9(a), is present in all packets. The Feedback header, shown in Figure 3.9(c), is optional and is included only in packets that carry feedback information. The extra headers, shown in Figure 3.9(b) are only present in the first and last packets of the sender for the connection. In the payload part of the packet, there might also be application-dependent headers. These headers belong to ATC and are usually used to store information about how to reconstruct ADUs before passing them back to the application. The ATC headers depend on the specific controller that is used and so they are not presented here, but are discussed in Chapter 4. In Appendix B, there is a detailed description of all the JTP packet fields.

The packet type field is a 5-bit mask field, where each bit corresponds to a different type of packet:

- If the **first bit** is set then it is a *feedback (ACK) packet* and the feedback headers are present.
- If the **second bit** is set then it is a *Connection Establishment (CE) packet*. CE packets cause a connection to be established between end hosts. The first packet of each connection has to be a CE packet. If this bit is set then the edge packet header is present. JTP strives to minimize redundant or duplicate transmissions and thus if the CE packet is perceived to be lost by the sender, the second CE packet sent carries new data just in case the first CE packet was not lost but just delayed in the network, or the ACK packet was lost. This optimization reduces the number of packets in the network but also makes it essential to include the id of the first packet in the transmission in each CE packet, as shown Figure 3.9(b).
- If the **third bit** is set then it is a *data packet (DATA)*. Any packet carrying application data should have this bit set.
- If the **fourth bit** is set then it is a *probe packet (PROBE)*. Probe packets are sent by the sender, when a feedback packet has not been received for some time. How much time should elapse before the sender sends a probe packet is indicated by the receiver in feedback packets.
- If the **fifth bit** is set then it is a *connection termination (CT) packet*. CT packets cause a connection to be terminated. Either side of the connection can send a CT packet to initiate the termination of a connection. For reasons similar to the ones explained for the CE packets, the edge packet header must be present in a CT packet to indicate the id of the last packet of the connection.

The fact that each bit of the packet type is a flag that can be set or not indicates that one packet can be of multiple types (e.g. the first packet of a connection has the CE flag set

and if it also carries application data, it has the DATA flag set as well.) A packet that has all the flags set is a special packet that is called a *connection reset (CR) packet* and is sent when one of the hosts wants to abnormally terminate a connection (i.e. without waiting for the other party to acknowledge the termination).

Note that this is an optimized version of JTP headers that only carry the necessary information. In our prototype implementation the JTP headers are slightly longer, mainly for debugging purposes.

The most important packets for a JTP transfer are the data and feedback packets for which we provide a more detailed description.

Data Packets

Data packets travel from the source to the destination of a JTP connection. In JTP, there are some novel fields in the basic header: minimum available rate (min rate), loss tolerance, energy budget and energy used.

• *Minimum available rate*: The available rate of a link, from a node to its neighbor, represents its current available *transmission* capacity—in JAVeLEN MAC, like a TDMA MAC, that available rate is determined by the current rate of unused (idle) time slots during which the neighbor is awake for reception; in CSMA/CA networks, a method similar to [LDJ04] can be used to estimate the available capacity. At each node visited the packet is stamped with the minimum available rate collected so far along the path of the JTP connection. If the link layer does not provide the available capacity JTP can estimate the available capacity by measuring the hop-by-hop delay similar to ATP [SAHS03]. As described in Chapter 6 the available rate is used by the flow controller at the destination to update the sending rate of the source. Note that due to retransmissions that may be required to get the packet to the next hop, a packet may consume more than one MAC-level transmission slot. So the available rate value must be normalized by the average number of MAC-level transmissions. Allowing multiple MAC-level transmissions can increase packet delay and also reduce the effective capacity that an application perceives [Lud00b]. As we will see later, by allowing

application requirements to dictate the number of MAC-level transmissions per link, JTP effectively gives applications control over the delay and effective capacity on every link.

The minimum available rate field addresses the goal of avoiding congestion losses. Since JAVeLEN provides a practically collision-free MAC layer, if the JTP flow controller ensures that it does not drive the available rate to zero, congestion-induced losses are avoided. Even if the underlying MAC does not provide collision-free access JTP's operation will not be affected since collisions would only increase the link loss experienced by packets, thus increasing the number of link-layer retransmissions per packet and effectively reducing the measured available capacity which in turn forces the sources to back off.

• *Loss tolerance*: A source node encodes the desired end-to-end loss tolerance in packet headers. Each node along the path precomputes the maximum number of transmission attempts to the next hop given the remaining length of the path (known from the node's view of the topology) and packet's loss tolerance. The packet is dropped if this predetermined maximum number of local attempts is exceeded, since exceeding that limit is more that what the application requires. As described in Chapter 5, before forwarding the packet the node updates the loss tolerance field so any left-over attempts (from the predetermined maximum number) do *not* get used downstream, thus reducing the variability in energy consumption across nodes along the path.

• *Energy budget and energy used*: The source initially assigns each packet an energy budget value based on the energy the network would *typically* expend to deliver the packet successfully. A packet is dropped whenever the energy used exceeds the energy budget of the packet. This approach provides an energy-conscious mechanism for dealing with routing loops (as opposed to the traditional hop-count TTL) and in conjunction with the loss tolerance field creates a sturdy way to manage the energy expenditure per packet. Important packets (ACK, CE, CT, PROBE) are assigned the maximum energy budget so as to ensure that if there is a path between the end hosts then they will reach the destination. This is done because these are important packets that if lost will affect the performance of the connection and also will be retransmitted anyway at a later time – so dropping them because

they exceeded the typical energy expenditure will not conserve any energy for the system.

The energy budget, energy used, and loss tolerance fields manage the expenditure of energy per packet. By limiting the effort of each node to successfully deliver a packet the loss tolerance field bounds the total transmission energy spent by each node. The energy budget on the other hand sets an upper bound on the energy that the whole network might expend to deliver a packet to the destination.

Feedback (ACK) Packets

JTP feedback packets carry acknowledgments as well as transfer parameters (transmission rate and energy budget information) from the receiver to the sender. The rate at which ACKs are fed back to the sender is regulated by the receiver based on the stability conditions of the path. For a stable path a minimum feedback rate is determined by the application based on its requirements—for example, an application with a more stringent delay requirement would require a higher feedback rate to achieve a more timely recovery of missing data. A lower feedback rate, if tolerated by the application, allows JTP to *aggregate* feedback information in a single packet, thus reducing feedback load.

Rate-based flow control systems, as with any networked control system [BHJ10], are vulnerable to the loss of feedback signals [ZBP01]. This is especially important in rate-based systems since the sender will keep transmitting at a constant rate until they receive a signal that the rate is not appropriate and should be adjusted. Imagine the case where there is congestion in the network and the source should reduce its rate to alleviate the condition. If the feedback packets that would notify the source that the network is congested are lost due to congestion for example, then the source will keep blasting the network with packets exacerbating the problem. In JTP the source proactively backs-off its transmission rate when a feedback packet has not been received on time; see Chapter 6, for more details on this mechanism. Furthermore, significant short-term variations in path conditions (available rate or energy used) would be detected by the path monitoring function at the receiver, thus triggering an early feedback.

In addition to reporting rate and energy information, a feedback packet carries both positive cumulative and selective negative acknowledgments. Each intermediate node on the path examines the SNACKs and retransmits missing packets if these packets are in the intermediate node cache—if they are, the node appropriately modifies the feedback packet so the sender is explicitly informed of such in-network retransmissions done on its behalf (cf. Chapter 5). Caching, and acknowledgments, are discussed in more detail in Chapter 5.

Figure 3.10 shows the path of JTP packets in the system — how they are passed from

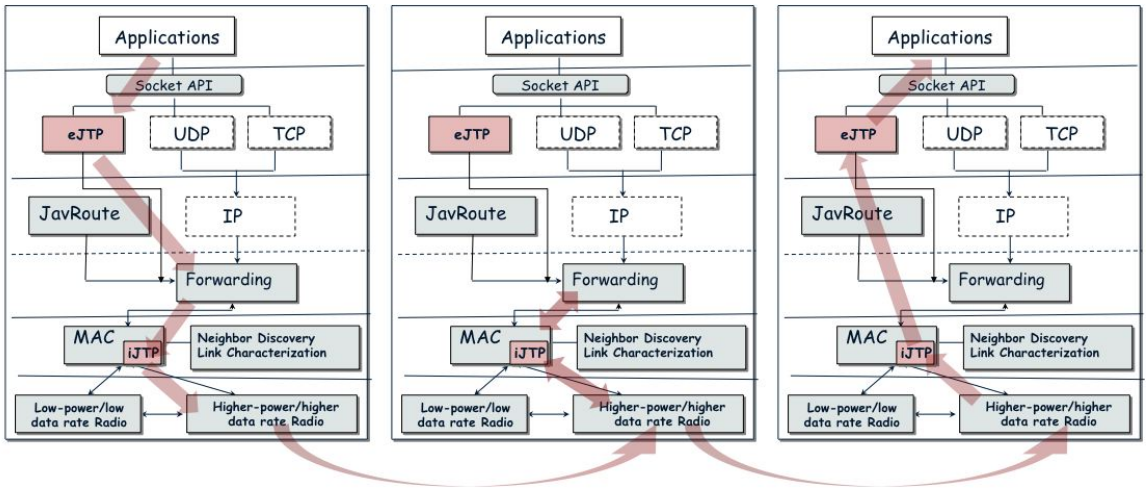


Figure 3.10: The path of a JTP packet through the network

module to module and from node to node. DATA packets are formed based on the ADUs given by the application to eJTP, while the rest of the packets originate directly at eJTP. JTP packets are sent down to the MAC. The rate at which DATA packets are sent is determined by the current transmission rate while ACK packets are sent based on the feedback rate and the stability of the path. At mid-path nodes, JTP packets only go through iJTP that is running in the MAC and then are forwarded to the next node. When the DATA packet reaches the destination, the original data is reconstructed at eJTP and is passed up to the application.

Figure 3.11 shows the lifetime of a typical one-way JTP connection. To establish a bi-directional connection, the destination can at any point initiate the establishment of the

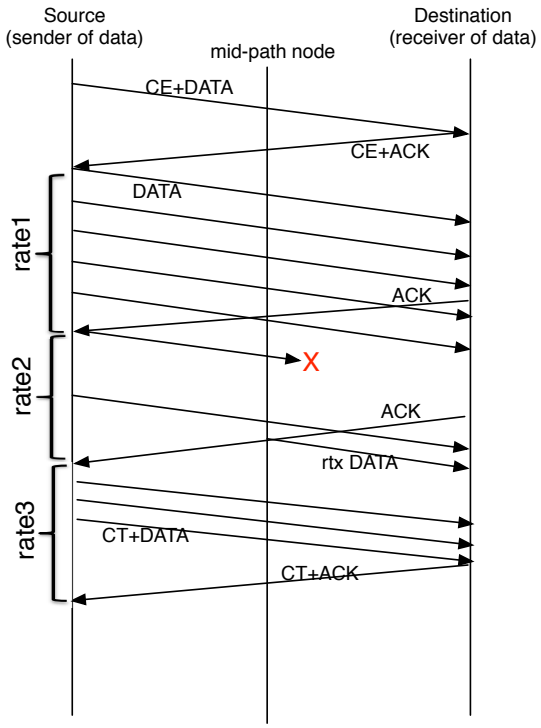


Figure 3.11: JTP connection diagram

connection by sending a CE packet. The rate at which data flows is constant and is set based on the information included in feedback packets. Losses are recovered as close to the destination as possible. Note that data can immediately start flowing minimizing the control overhead of the connection.

3.3 Conclusion

In this Chapter we presented in detail the design of JTP. JTP has two main goals: (1) to support a diverse set of applications while (2) minimizing the energy spent by reducing the total number of packet transmissions.

To achieve its first objective, JTP has a separate module, the ATC module, that is placed between the applications and the mechanisms for transferring data over the network. ATC exposes well-defined APIs to both the applications and the rest of JTP, enabling different

ATC modules to be operate simultaneously. Each ATC module is customized to make transfer decisions for a different type of applications. For example, the ATC module for bulk transfers enforces full reliability, while an ATC module for streaming data strives to minimize delivery delay.

Although JTP preserves the end-to-end principle, it also performs hop-by-hop operations in order to improve performance and react fast to network changes, avoiding unnecessary packet transmissions. JTP is divided into eJTP, which is responsible for the functionality at the end points (source and destination), and iJTP, which performs the in-network processing for JTP. Since iJTP is in the fast path of packet forwarding, it is kept simple while more complex functions are implemented at eJTP.

Chapter 4

Application Dependent Transfer Control

JTP is envisioned to support a wide range of applications. In order to achieve this goal, it is essential that each data transfer task (e.g. congestion control, routing, error recovery) be properly designed to decouple functions that directly affect the performance of the application and functions that only have to do with the underlying network. Such separation of the transport protocol's mechanisms allows it to tailor its policies to accommodate different application requirements, while maintaining the commonality of interacting with the lower layers of the system.

It was recognized early on that the stream-byte approach of TCP is not appropriate for all applications. RDP [VHS84] was proposed in 1984 as a reliable transport protocol for remote debugging and application loading. The authors decided to go with a packet-based approach since it would simplify the protocol design and implementation as well as allow for out-of-order delivery of packets. VMTP [Che88] is another packet-based transport protocol specifically designed to support the transaction model of communication. Clark further formalizes this observation in [CT90a]. In this work, Clark points out that not all applications have the same requirements in terms of how and when data should be delivered to them. In time-sensitive, realtime applications (such as VoIP or video streaming), the application can deal with lost data but it is important to receive new data as soon as they arrive as long as they are combined and presented in a form that the application can parse.

From the legacy protocols only UDP respects the boundaries of the data as it is given by the application (i.e. application framing), while TCP transforms all data into a stream of bytes and thus loses the original framing. TCP has no other option than to deliver a stream of in-order bytes to the application. Applications that care to maintain the original framing of the data, have to use UDP, which on the other hand does not provide any form of reliability. Using UDP, applications are forced to add the desired reliability at the application layer which can be very inefficient. ITP [RBS00], a protocol designed for image transfers, is another good example of a variant that provides a reliable transfer but still maintains the framing of the application, which allows out of order delivery. An application using ITP, allows the user to slowly start seeing different parts of the image as they arrive, without having to wait for the whole image to be downloaded, or for the segments to arrive in order. In this chapter we discuss how JTP is designed to support different types of applications – striving to satisfy the idiosyncrasies of each while still being efficient in the use of network resources.

4.1 eJTP: Application Transfer Controller (ATC)

In order to address this challenge, JTP's design introduces a module that implements all the application-dependent functionalities called *Application Transfer Controller (ATC)*. This module is responsible for influencing specific QoS parameters that the network supports based on the application's needs.

JTP provides a set of adjustable knobs to the application in order to optimize the network performance from its point of view. Given that JTP is designed to operate within an ad hoc network setting, the knobs provided to the application are:

- **Delay:** The applications can specify whether the data to be transmitted is time sensitive, for example data in a streaming application are not useful if they are delivered after their playback time, while in a file transfer the data packets are useful independent of when they arrive.

- **Reliability:** The application can specify what are the reliability requirements it has. For example, in a streaming application, limited losses can be tolerated without compromising the usefulness of the application.

- **In–order delivery:** Just like RDP [VHS84], JTP provides the capability to the application to control whether it can handle out–of–order packets or not.

- **Transfer size:** Just like VMTP [Che88], JTP tries to keep simple transactions (e.g. a flow that only has one packet) to a minimal set of packet exchanges. The application gets to specify what is the transfer size for the specific flow. Based on that the ATC module can make decisions about whether to piggy-back data along with the connection establishment, and termination packets. For example if the application is planning to transmit only one frame then JTP can send only one packet that is a CE, CT and data packet, and based on the reliability requirements either wait for an acknowledgment or immediately tear down the connection.

- **One or two-way communication:** This instructs JTP whether it should establish a two-way connection, allowing data transfer both ways, or whether this is a one-way transfer and thus there is no need to exchange extra data for setting up and tearing down the other direction.

The interface between an application and JTP, allows the application to set each one of the above parameters by using a generic, tunable ATC. Except from that ATC, JTP also supports the use of highly specialized ATC modules that set the above parameters for the application to appropriate values and thus removing the burden from the application to fine tune the transfer. Figure 4.1 shows how the applications interact with ATCs. Each application is linked to an ATC and each ATC is linked to eJTP's core modules. Furthermore, the eJTP architecture aims at facilitating the insertion of new modules by providing a standard API that new modules need to implement. Although it is possible for an application to want different data to be handled by different ATCs, and the JTP architecture allows such

Table 4.1: Example of application requirements

Example Application	**Delay**	**Reliability**	**In–order**	**Transfer size**	**one/two-way**
Streaming Video	tight	$< 100\%$	no	large	either
Interactive	tight	$\leq 100\%$	yes	small	two-way
Periodic Transmissions	moderate	$< 100\%$	N/A	1 frame	one-way

a possibility, the current implementation allows only one ATC per connection and thus applications that need to use more than one ATC, should create multiple connections.

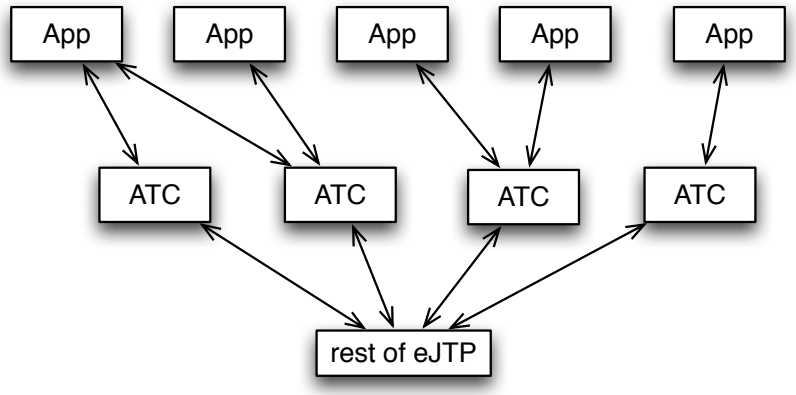


Figure 4.1: JTP can support multiple ATC modules and each application is free to choose one or more.

In Table 4.1 we list some example applications and how they would set the above set of parameters.

JTP's API for communicating with the applications is summarized in Table 4.1. For presentation clarity we omit the details for each call and just describe the high level functionality. For more details the reader is referred to Appendix A.1.

Before listing all the mechanisms that ATC is responsible for, it is worth mentioning that JTP supports the parallel operation of multiple ATCs. All ATCs support some basic functionalities and have common interfaces with the rest of the eJTP modules. Here we do provide a succinct list of the ATC responsibilities both at the sender and at the receiver but we omit the implementation details. For a complete list of functions that each ATC

Table 4.2: JTP Application API

Function Call	**summary of operation**
create()	This function is the equivalent of the socket() call in Linux; it creates a new connection and returns a descriptor of the created connection.
setconnopt()	This function is used by the application in order to set specific parameters. If an option is not recognized by JTP, the set option command is forwarded to the ATC module in use.
getconnopt()	This function is used by the application in order to get the value of specific parameters. If an option is not recognized by JTP, the get option command is forwarded to the ATC module in use.
connect()	This function is similar to the connect() call in Linux. It is used when the application wants to initiate a connection with a remote host. A call to the create() function should precede this call.
bind()	This function is similar to the bind() call in Linux.
listen()	This function is similar to the listen() call in Linux.
accept()	This function is similar to the accept() call in Linux.
send()	This function is called by the application in order to send an Application Data Unit (ADU) to the destination. If this function is invoked on a connection that has not yet been established yet, i.e. there was no connect() call preceding this, then this function operates similarly to the sendto() function in Linux.
receive()	This function is similar to the receive() function in Linux. If this function is not called on an established connection then the semantics are similar to the receivefrom() function in Linux.
close()	This function is similar to the close() function in Linux and it can be called only for connections that were created through the create() call.

module is responsible for implementing please see Appendix A.2.

At the source, ATC is responsible for the following tasks:

- It is responsible for **placing** received **ADUs** in **JTP packets**. For some applications it might make sense to pack multiple ADUs into the same packet, for others it might be more beneficial to have each ADU in a separate packet, etc. ATC is basically responsible for fragmenting (or aggregating) and reassembling(or separating) ADUs.
- Based on the application's tolerance to losses, it **assigns a loss tolerance** level to each packet.
- It is responsible for deciding:
  - whether to piggy-back data with the CE packet
  - whether this is just a data transfer and the connection should be closed after the data is sent

At the destination, ATC implements the following functionalities:

- It reconstructs ADUs from the received packets, and delivers them to the application **in or out of order**, based on the application's requirement.
- It decides **which** packets to **acknowledge** as received, and **which** of the **lost packets** should be **recovered** by the network. Many times the application doesn't care about receiving all the packets, but there might be specific packets that are important. In a video streaming application for example, ADUs that have missed their playback time are not useful to the application, and from the ADUs that are still useful, some (those that contain key frames) are more important than others in terms of the perceived video quality.
- It decides **when** to send **feedback packets** to request for recovery of lost packets.
- Based on the network statistics gathered, the ATC module assigns the energy budget for each packet.

- It instructs the source about how much data it is allowed to send (i.e. the receiver advertised window).

In order to complete our JTP implementation we designed and implemented two different ATCs; the *Reliable ATC* and the *Voice ATC*. The *Reliable ATC* supports 100% reliable transfer and we used it to compare against other protocols since most other transport protocols only support full reliability. The *Voice ATC* is a customized ATC, specifically designed to support voice streaming. We implemented this ATC to explore the potentials of JTP in conserving energy while still providing good service to the application.

4.2 Reliable ATC

In our JTP implementation the Reliable ATC is the default module used by any application and it is the module that has been used in all the experiments presented in Chapter 7.

4.2.1 QoS Parameters

The Reliable ATC, provides a full reliability service to applications that are not concerned with transfer delays. It uses the following parameters:

- **Delay**: infinite, i.e. there are is no delivery deadline for the corresponding JTP packets — as long as the packet is delivered at the destination it is always considered valid.

- **Reliability**: 100%,i.e. this module requires that all packets are delivered.

- **In-order delivery**: the Reliable ATC module has an option that the application can set, for whether it wants in-order or out-of-order ADU delivery. The option name is $JTP\_IN\_ORDER\_DELIVERY$ and the application can set it to:

  - 0: if the application wants ADUs to be be delivered as soon as they are received by JTP, or

  - $\neq 0$ (default): if the application requires in order delivery of ADUs. This is also the default setting since most existing applications that require full reliability are usually designed for TCP and thus require in order delivery to operate correctly.

- **Transfer size**: This is an implicit tunable parameter as well. If the application wants to send only a single ADU, then it should call the Send function of the API. If the application wants to send multiple ADUs, then it first invokes the Connect function and then the Send function.

- **One or two-way communication**: For the purpose of the current implementation we assume only one-way communication, since most applications that are used in JAVeLEN have information flowing only one way.

4.2.2 ADU Packing

As discussed previously in this chapter, the ATC module at the source is responsible for providing the payload in JTP packets while at the receiver side, it is responsible for reconstructing the original ADUs before passing them to the application. Reliable ATC tries to fill each JTP packet as much as possible with application data to minimize the total number of packets that are needed to complete the transfer. Each packet transmission comes with a constant overhead for processing the packet (e.g. processing the packet in the system, turning on the radio for transmission/reception) that is not dependent on the packet size or at least not linearly dependent; i.e. the processing and transmission of one packet will use less energy than that of two packets that are half the size [EBW02]. Since there are no other QoS requirements (delay, etc.), Reliable ATC will start adding data into packets until the packet is of maximum size. This is similar to how TCP works, with the only difference that it maintains the ADU framing. Also this module will process the ADUs in the same order as they are received from the application.

In order for the receiver to be able to reconstruct the original ADU, ATC adds a header before each ADU in a packet. The ATC headers are presented in Figure 4.2.

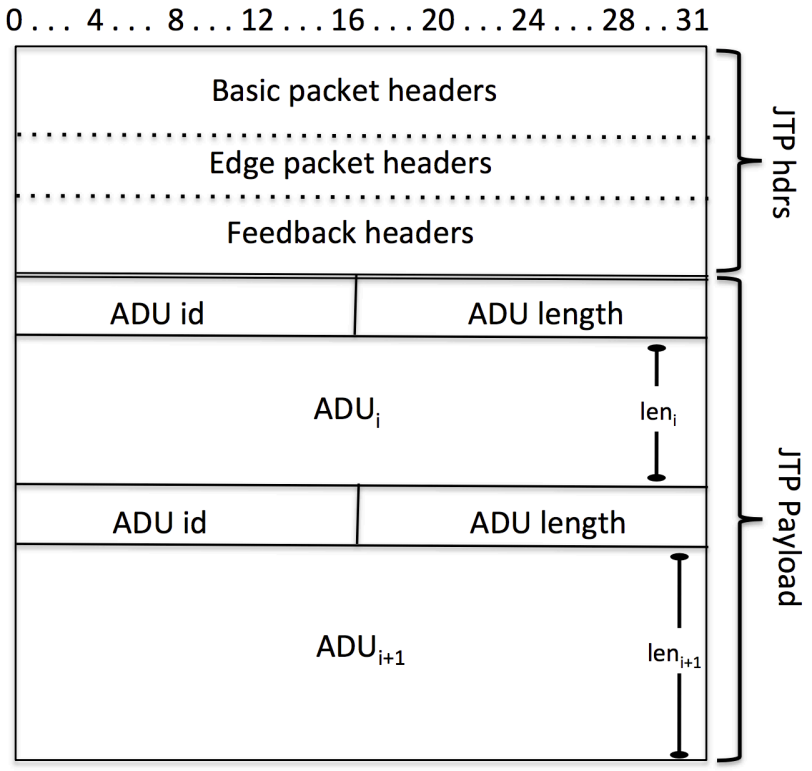


Figure 4.2: Reliable ATC module headers that precede the actual ADU

Multiple ADUs can be placed in one JTP packet. A packet might contain only a part of an ADU. The packet that contains the rest of the ADU will also include the ATC headers before the actual data, to ensure correct reassembly. There are only two fields in the Reliable ATC headers: the *ADU id* of the ADU that follows, and its *ADU length*. The assumption is that a new ADU will not be sent before the previous one is fully sent, i.e. the beginning of a new ADU signifies the end of the previous ADU. This also implies that an ADU can be split only in consecutive packets. This will always guarantee correct reassembly of ADUs. As the receiver gets JTP packets, it aligns them in increasing order of the JTP packetID and reconstructs the ADUs for which it has all consecutive JTP packets, from the first occurrence of the specific ADU id all the way to the next ADU id.

4.2.3 Loss Recovery

The Reliable ATC provides 100% reliability to the applications, which means that eventually all JTP packets have to be delivered for the transfer to be successful. Given that there are no delay requirements imposed by the application, the frequency at which the receiver requests for recovery of lost packets, depends only on the condition of the network. Every time that the network characteristics change significantly, causing the receiver to send a feedback packet, recovery information is all sent along. Also as described in Chapter 5 the cache size also affects the frequency with which lost packets should be requested. Although more details are presented in Chapter 5, the high level intuition is that as new packets arrive, old packets are evicted from the caches and thus if the receiver waits a long time before requesting a missing packet, the packet might have been evicted from all caches, forcing the recovery to happen from the source, which wastes a lot of energy.

4.3 Voice ATC

4.3.1 QoS Parameters

Unlike the Reliable ATC, an ATC that is designed to support Voice applications, it doesn't need to provide full reliability to the applications but it should try and respect the delay requirements of the applications. This is because if an ADU misses its playback time at the destination, it will be dropped anyway, wasting all the energy spent to transfer the data.

- **Delay**: the Voice ATC module has an option that the application can set, in order to specify the maximum delay an ADU can experience. The option name is JTP_MAX_DELAY and the application can set it using the set option function. Valid values for this option are all positive integers. A value of 0 signifies *infinite* delay. A value of $0xFFFF$ specifies that the application defers the setting of this parameter to ATC. The value is in milliseconds. By default the Voice ATC sets the maximum delay for the connection to be twice the current estimation of the Round Trip Time (RTT). It is best for the applications to let ATC decide on the right value for the maximum delay since an effective value depends on the network conditions and the value and jitter of RTT that is information that the ATC has. The application can use the JTP_AppIntfGetConnOpt() to read it's latest value from ATC. The deadline for each ADU is set by the ATC by adding the maximum delay value to the time the ADU is received from the ATC.

- **Reliability**: the Voice ATC module has an option that the application can set, in order to specify its tolerance to lost ADUs, either due to network losses, or due to missed deadlines. The option name is $JTP\_LOSS\_TOLERANCE$ and the application can set it using the set option function. Valid values for this option are integers from 0 to 100, signifying the percentage of lost packets that the application can handle. The default value is 5.

- **In-order delivery**: in a voice application delivering ADUs as soon as they are re-

ceived is very important since it is more important to make the ADU deadline rather than delivering everything in order. The Voice ATC, delivers an ADU as soon as it is completely received, even if it is out of order, i.e. it only supports out-of-order delivery.

- **Transfer size**: the voice ATC assumes that there will be a stream of packets and thus invokes the connect function that establishes a connection before any application data is exchanged. This way the voice ATC has a chance to estimate the RTT in order to set the maximum delay in subsequent JTP packets.

4.3.2 ADU Packing

The Voice ATC uses similar ADU packing as the Reliable ATC. When it is time for JTP to send out a new packet, it will pack as many ADUs as possible and send the packet. The only difference is that as long as there is at least one complete ADU included in the packet, it will not fragment ADUs; i.e. each JTP packet only includes complete ADUs. The only situation where the Voice ATC fragments an ADU is if the JTP packet size is smaller than the ADU size in which case it is mandatory to fragment. This allows JTP to drop JTP packets in the middle of the network without wasting energy delivering only parts of an ADU. This does not apply to the Reliable ATC module since it requires 100% reliability and thus all the JTP packets have to be delivered. The deadline of each JTP packet is set to the earliest deadline among the deadlines of the ADUs that are packed in the packet. The ATC headers are the same as the ones used for the reliable ATC shown in Figure 4.2.

4.3.3 Loss Recovery

The Voice ATC provides reliability that is typically less than 100% though it has to stay within some bounds. This implies that although not all packets need to be delivered in order to satisfy the requirements of the application, if the packet loss is too high, JTP still needs to act to recover some of the lost packets if possible. If data is dropped because their deadline is missed, then there is no point in trying to recover it and thus there is not

much the receiver can do. Unlike the Reliable ATC module, the ADUs in the Voice ATC have delay bounds and thus should be requested for retransmission only if their playback deadline has not expired. The way the Voice ATC decides which packets to request for retransmission and when, is as follows:

1. The Voice ATC maintains a running average of the most recent packet loss percentage. This includes all packet drops whether they happened because of transmission error or because of expired deadlines.

2. When the packet loss exceeds the loss tolerance of the application, JTP immediately sends a feedback packet.

3. In the feedback packet JTP requests for retransmission packets that have not yet expired. JTP is trying to ensure that the packets that are requested contain ADUs that will not have expired by the time they are received by the destination. In order to achieve this, JTP computes the expiration time of lost packets based on the expiration time of received packets, and it accounts for the round trip time when it decides which of the lost packets are to be retransmitted.

4.3.4 Simulation Results

For this thesis we just run some very basic experiments to evaluate whether the Voice ATC, which allows for some packet loss, can achieve acceptable voice quality while conserving energy.

We used the OPNET [opn] simulator for our experiments. We studied the behavior using linear topologies of growing sizes. The main metric that we used to assess the performance of voice traffic carried over JAVeLEN is *PESQ mean opinion score (MOS)*.

This metric assesses the quality of delivering a real audio trace, taking into account the different phases of coding, transporting over the network, decoding and playing back the audio at the destination. This MOS metric is standardized by ITU [HWW04] and captures distortion due to the coding scheme used and loss of voice frames in the network

or during playback as frames may miss their playback time due to jitter. This MOS value ranges between 1 (poor speech quality) and 4.5 (excellent speech quality). More specifically, PESQ-MOS values higher than 3.6 indicate good speech quality; PESQ-MOS values between 3.2-3.6 indicate acceptable quality; while values lower than 3.2 indicate unacceptable or bad speech quality [MTK02, HWW04].

Voice is assumed to be sampled at a certain rate (8 Kbps using G.729 coding [MTK02]), and 10-ms voice frames are packetized in JTP packets to be sent over JAVeLEN.

For the purpose of this comparison we used three different versions of JTP:

- **JTP**0% **LT**: This is JTP using the Voice ATC module but not allowing for any losses.
- **JTP**10% **LT**: This is JTP using the Voice ATC module but allowing for up to 10% of the packets to be lost.
- **UDP**: This is basically JTP with no reliability, i.e. there are not retransmissions after a packet is lost in transit.

Figure 4.3 shows the performance of G.729 voice flows. For the purpose of this experiment there is an ADU produced every 10ms. We observe that good speech quality can be achieved by leveraging the in-network recovery of JTP while delivering less data and thus conserving energy. .

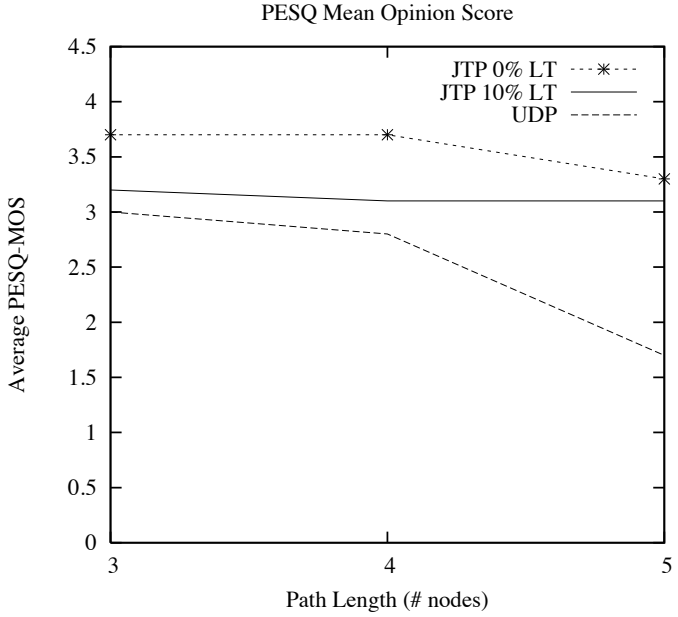


Figure 4.3: Performance of G.729 voice flows. Good speech quality can be achieved by leveraging the in-network recovery of JTP while delivering less data.

Chapter 5

Error Control

One of the most important challenges when designing protocols for wireless networks, is the high Bit Error Rate (BER) of the communication channel. Although the actual value of BER is influenced by a number of factors such as transmission power, modulation scheme and interference, the average values are significantly higher [GSK94] than those of wired networks. The unreliability at the physical layer propagates up the stack, deteriorating the performance of many of the widely used network protocols that have been designed for wired networks where packet losses due to channel errors are rare [FML02, AS04, HV02, BPSK97].

Protocols that are envisioned to operate in a wireless setting, should be specifically designed to handle random packet losses and high BERs. Error recovery in wireless networks has been an active area of research for many years [PW72, Moo05]. There are mainly two different approaches to cope with the high BER that is encountered in wireless communication systems:

1. the proactive approach that attempts to ensure that there is enough information at the receiver to correctly decode the packet even if some bits are corrupted,

2. the reactive approach that retransmits packets that were corrupted and thus dropped at the receiver.

Most systems deploy both of these approaches, usually in different layers.

Forward Error Correction (FEC) is the most common proactive approach that is deployed to enhance the decoding capabilities of the receiver in the face of corrupted bits. The main idea is that the sender encodes the message using error-correcting codes that includes redundant information to enable the receiver to recover from individual bit errors in a packet [PW72]. In wireless communication networks where the quality of the channel changes significantly over time, there is the need for adapting codes to match the current state of the channel. In some networks, e.g. GSM networks, a predefined set of channel codes is provided and then the sender can choose which one to use based on the conditions of the channel. Ahn et al. propose dynamically adapting the strength of FEC [AHH05], to match the constantly changing wireless channel status. Researchers [BYJK07, Hav99] have also studied the energy efficiency of different error-correcting codes, noting that when evaluating different protocols in terms of energy, the total energy consumed by the device should be taken into account and not only that consumed by the bits that are being transmitted.

Another approach that researchers have been looking into for addressing the high BER in wireless communications is *network coding*, where instead of focusing on the transmission of a single packet between two hosts, this technique performs coding over several packets and hosts. The shared communication medium of wireless networks is the ideal platform for deploying network coding techniques. Katti et al. [KRH$^+$06] proposed to use XOR to encode/decode multiple packets. Silva et al. [SKK07] looked at the performance of random linear network coding for error control from a theoretical perspective, while Park et al. [PLS$^+$06] have studied the performance of network coding in ad hoc networks, especially for multicast traffic.

To improve the network efficiency in wireless communications where the quality of the channel is varying significantly over time, link adaptation techniques other than adjusting the coding used, have been proposed. Researchers have proposed to adapt the channel rate in an effort to improve the channel conditions at the expense of transmission speed [HVB01, SKSK02]. Another interesting approach that has been proposed is to control the transmission power [Chi05, EKCD00].

In an energy-constrained environment, it is not straightforward how much effort should be put in, in delivering one packet, the challenge of employing these techniques is even greater since most of these schemes require higher energy consumption. One should decide which losses are worth recovering from, and how to recover in the most energy efficient way.

JAVeLEN employs multiple of the above mentioned techniques for improving the BER, but allows JTP to control them on a per packet basis. Each packet is accompanied by a transmission profile that instructs the JAVeLEN radio how each packet should be transmitted. The transmission profile among other things controls the power to be used, which coding scheme to use and how many times the packet should be retransmitted until it is successfully delivered to the next hop. Although JAVeLEN provides great flexibility to JTP to fine tune many transmission parameters, in this thesis we explore the impact of adjusting the ARQ, based on the reliability requirements of the application.

Even if all the link-layer mechanisms are optimally configured, losses are unavoidable in a mobile, wireless setting. JTP is trying to recover only from necessary losses and as close to the receiver as possible in order to reduce the packet transmissions that are necessary for the application to be useful.

In summary, JTP uses *dynamic in-network retransmissions, in-network caching and Selective packet Negative ACKnowledgments (SNACKs)* to overcome this challenge. This chapter explains in detail these mechanisms of JTP.

5.1 High level Analysis of JTP benefits

In order to study the benefits of JTP's error control mechanisms, we present an average case analysis of the energy consumption of JTP, TCP, and UDP. This is a high-level analysis that provides a proof of concept of JTP's design, by modeling and comparing the error handling of JTP, TCP, and UDP. We examine the energy consumption of a transmission of k packets over a path of H hops. To make the analysis tractable, we make the following assumptions:

- We consider losses due to link errors, and not because of congestion. Although this is a best-case scenario for TCP, it is an ideal representation of JAVeLEN that strives to eliminate any losses due to congestion both at the link-layer (TDMA-based scheme) and at the transport layer with JTP.

- We do not consider the energy consumption of ACKs. ACK packets are much smaller than the data ones and thus the energy consumed can be ignored.

- The probability of a packet being dropped on a link is the same for all links and equal to p. This is for ease of computation and does not affect the result; p can be considered as the upper bound on link loss.

- We assume independence between different transmissions

- The packet size is fixed

- The energy consumption for the transmission of one packet is the same over any hop and equal to $\mathcal{E}_{base}$

- JTP has infinite caches, i.e. there are no cache misses. This approximates the best-case scenario for JTP, with the worst-case being the results of TCP.

5.1.1 Performance Metrics

We compare the error handling of three transport protocols, TCP, UDP, JTP, in terms of the following performance metrics:

- **Expected Energy Consumption** $E[\mathcal{E}_{tot}]$. This is the average energy that is spent by the system to transfer k packets given by the applications, over a path of H hops when the loss probability for every hop/link is p.

- **End-to-end loss tolerance** l_{e2e}. We say that a transport protocol satisfies a loss tolerance of $l_{e2e} \in [0, 1]$, if it *guarantees* to deliver on average $(1 - l_{e2e}) \times k$ packets. The value of l_{e2e} is independent of $H, k, \mathcal{E}_{base}$, and p.

- **Expected Energy Consumption per bit delivered** $\frac{\partial E[\mathcal{E}_{tot}]}{\partial k_s}$ where k_s denotes the number of successfully delivered packets. This measures the average energy consumption per successfully transmitted bit.

5.1.2 Modeling End-to-end Error Handling of TCP

In TCP if the application gives k packets to the transport layer, then it is guaranteed that all k packets will reach the receiver, i.e. $k_s = k$. The probability, P_{e2es} that a packet will be successfully transmitted through all the H hops is

$$P_{e2es} = (1-p)^H$$

So the probability that a packet is lost is

$$P_{e2el} = 1 - P_{e2es} = 1 - (1-p)^H$$

In TCP every time a packet is dropped, the sender retransmits the packet. If we consider each time a sender transmits a packet as a Bernoulli trial with probability of success P_{e2es}, then the average number of retransmissions per packet is the average number of Bernoulli trials until the first success. Let X be a random variable that denotes the number of Bernoulli trials until the first success (including the successful transmission). We know that X follows a geometric distribution with expected value:

$$E[X] = \frac{1}{P_{e2es}}$$

Let $K \geq k$ be a random variable representing the total number of packets sent by the sender, i.e. the total number of Bernoulli trials until the sender successfully transmits all k packets.

$$E[K] = kE[X] = \frac{k}{P_{e2es}}$$

Out of the K packets, $(K-k)$ are packets that were lost. Now given that a packet is lost we would like to compute the expected number of hops it traversed before it was dropped. Let O_l be a random variable that denotes the number of one-hop transmissions for a packet that is eventually lost.

$$
\begin{aligned}
E[O_l] &= \sum_{i=1}^{H} i \times Pr[\text{packet lost at hop } i | \text{packet is lost}] \\
&= \sum_{i=1}^{H} i \times \frac{Pr[\{\text{packet lost at hop } i\} \cap \{\text{packet is lost}\}]}{Pr[\text{packet is lost}]} \\
&= \sum_{i=1}^{H} i \times \frac{(1-p)^{i-1} \times p}{P_{e2el}} \\
&= \frac{p}{1-(1-p)^H} \times \sum_{i=1}^{H} i \times (1-p)^{i-1}
\end{aligned}
$$

Let E_{tot}^{TCP} be the random variable that represents the total energy spent for a TCP transfer of k packets.

$$
\begin{aligned}
E[\mathcal{E}_{tot}^{TCP}] &= \mathcal{E}_{base} \times k \times H + \mathcal{E}_{base} \times (K-k) \times E[O_l] \qquad (5.1) \\
&= \mathcal{E}_{base} \times \left(k \times H + \left(\frac{k}{(1-p)^H} - k \right) \times E[O_l] \right) \\
&= \mathcal{E}_{base} \times k \times \left(H + \frac{1-(1-p)^H}{(1-p)^H} \times \frac{p}{1-(1-p)^H} \times \sum_{i=1}^{H} i \times (1-p)^{i-1} \right) \\
&= \mathcal{E}_{base} \times k \times \left(H + \frac{p}{(1-p)^H} \times \sum_{i=1}^{H} i \times (1-p)^{i-1} \right)
\end{aligned}
$$

[1] Since all the packets are guaranteed to be delivered when we use TCP, the loss tolerance is 0, i.e.

$$l_{e2e}^{TCP} = 0$$

The average energy consumption per bit delivered is given by:

$$\frac{\partial E[\mathcal{E}_{tot}^{TCP}]}{\partial k} = \mathcal{E}_{base} \times \left(H + \frac{p}{(1-p)^H} \times \sum_{i=1}^{H} i \times (1-p)^{i-1} \right)$$

[1] Note that $\sum_{i=1}^{H} i \times (1-p)^{i-1} = \frac{1}{p^2} - \frac{(H+1)(1-p)^H}{p} - \frac{(1-p)^{H+1}}{p^2}$.

5.1.3 UDP Model

In UDP there are no retransmissions. Each packet is sent only once. Let K_s be a random variable representing the number of packets that were successfully delivered.

$$E[K_s] = k \times P_{e2es} = k \times (1-p)^H$$

The rest of the packets were dropped somewhere in the network. Let U be the random variable representing the number of one-hop transmissions that happened during a UDP transfer of k packets.

$$\begin{aligned} E[U] &= E[K_s] \times H + (k - E[K_s]) \times E[O_l] \\ &= k(1-p)^H \times H + k\big(1-(1-p)^H\big) \times \frac{p}{1-(1-p)^H} \times \sum_{i=1}^{H} i \times (1-p)^{i-1} \\ &= k \times \Big((1-p)^H \times H + p \sum_{i=1}^{H} i(1-p)^{i-1}\Big) \end{aligned}$$

Using the estimation of one-hop transmissions we can estimate the expected value for the total energy spent by UDP:

$$E[\mathcal{E}_{tot}^{UDP}] = \mathcal{E}_{base} \times E[U] \tag{5.2}$$

$$= \mathcal{E}_{base} \times k \times \Big((1-p)^H \times H + p \sum_{i=1}^{H} i(1-p)^{i-1}\Big) \tag{5.3}$$

Since effectively all the packets during a UDP transfer could be lost, the loss tolerance of UDP is 1, i.e.

$$l_{e2e}^{UDP} = 1$$

The average energy consumption per bit delivered is given by:

$$\frac{\partial E[\mathcal{E}_{tot}^{UDP}]}{\partial k_s} = \frac{\mathcal{E}_{base} \times \Big((1-p)^H \times H + p \sum_{i=1}^{H} i(1-p)^{i-1}\Big)}{(1-p)^H}$$

5.1.4 JTP Error Control Model

For this analysis we assume that error recovery is done using hop-by-hop retransmissions and not through other mechanisms such as FEC or modulation. In JTP every packet will

also have an energy budget $\mathcal{E}_b$ associated with it. Every time the packet is transmitted over one hop, the energy budget is updated to be $\mathcal{E}_b = \mathcal{E}_b - \mathcal{E}_{base}$. For simplicity, we assume that $\mathcal{E}_b$ is an integer multiple of $\mathcal{E}_{base}$. When $\mathcal{E}_b$ reaches zero the packet is dropped. Based on the assumption that $\mathcal{E}_{base}$ is common for all the hops, let $m = \frac{\mathcal{E}_b}{\mathcal{E}_{base}}$ be the total number of times a packet can be transmitted. Since $\mathcal{E}_{base}$ is considered constant for the purpose of this analysis we are going to consider m in our analysis and not $\mathcal{E}_b$.

We know that TCP has a loss tolerance of zero and UDP has a loss tolerance of one. JTP attempts to bridge the gap between these two extremes by allowing the application to specify $l_{e2e} \in [0, 1]$. JTP achieves this by using the energy budget, $\mathcal{E}_b$, to influence how many packets are expected to be delivered. For a packet to be able to reach the destination, it needs at least $m = H$. In this case JTP is equivalent to UDP, since it effectively allows for only one transmission per hop to successfully reach the destination just like UDP; $l_{e2e} = 1$. If $\mathcal{E}_{tot}^{JTP}(k, l_{e2e})$ is the total energy consumed for the transmission of k packets using JTP for a loss tolerance of l_{e2e} then

$$E[\mathcal{E}_{tot}^{JTP}(k,1)] = E[\mathcal{E}_{tot}^{UDP}(k)] \tag{5.4}$$

$$= \mathcal{E}_{base} \times k \times \left[(1-p)^H \times H + p \sum_{i=1}^{H} i(1-p)^{i-1}\right] \tag{5.5}$$

Note that it is *not* true that if we set $m = \infty$ then JTP is the same as TCP. Although JTP would guarantee the same loss tolerance as TCP, i.e. $l_{e2e} = 0$, the retransmissions in JTP are within the network, i.e. hop-by-hop from the caches and not end-to-end.

Let's calculate the average energy consumption for JTP in the case of $m = \infty$. Each time a packet is transmitted over one hop, there is a probability p that the packet is lost. Since $m = \infty$, the packet will keep getting retransmitted until it is successfully received. Let X be a random variable denoting the number of times a packet needs to be transmitted over a link until it is successfully received. We know that

$$E[X] = \frac{1}{1-p}$$

So if $\mathcal{M}$ denotes the total number of one-hop transmissions of a packet until it reaches the

receiver, we have

$$E[\mathcal{M}] = H \times \frac{1}{1-p}$$

In this case, where $l_{e2e} = 0$:

$$E[\mathcal{E}_{tot}^{JTP}(k,0)] \quad = \quad \mathcal{E}_{base} \times k \times \frac{H}{1-p}$$

Let's now assume that an application has a loss tolerance of $l_{e2e}, l_{e2e} \in [0,1]$. That means that at most $l_{e2e} \times k$ packets can be lost. One way to achieve this is to set m to ∞ for $(1-l_{e2e}) \times k$ packets and to the minimum value (i.e. $m = H$) for the rest of the packets. That means that the average energy consumption is

$$E[\mathcal{E}_{tot}^{JTP}(k,l_{e2e})] \quad = \quad E[\mathcal{E}_{tot}^{JTP}(l_{e2e} \times k,1)] + E[\mathcal{E}_{tot}^{JTP}(k \times (1-l_{e2e}),0)]$$

We now provide an average case analysis when $m \in [H,\infty)$. Let's assume that all the packets have the same m. In JTP if a node determines that the energy left for a packet is not enough to make it to the receiver the packet is dropped. For example, let's consider the case when a packet i is at node j, that means that it is $H-j$ hops away from the receiver. Let m_i be the number of transmissions that this packet has left. Then $m_i < H-j$, then this packet is dropped, since the minimum energy required to reach the destination is more than the current energy budget of the packet. Let's split m into two parts: m_b equals to the number of hops H that a packet must traverse and it is the minimum m that can be assigned (i.e. $m_b = H$), and $m_e = m - m_b$ is the extra times a packet can be retransmitted. If a packet is lost more than m_e times, then the packet should be dropped since it will never reach the destination. So we can view this as a coin-toss problem: Given the probability of observing a head is p, we will stop the experiment either when we observe (m_e+1) heads (analogously the packet is dropped as it exceeded its allowable number of retransmissions) or if we observe H tails (analogously the packet has successfully traversed all H hops).
Let's calculate the probability that a packet is dropped. The packet can be dropped on the first hop, or on the second hop, or on the third etc. For a packet to be dropped, that means it failed (m_e+1) times. Then if we consider each transmission as a Bernoulli trial with probability of failure p, then the result of the last trial must be a failure.

$$
\begin{aligned}
Pr[\text{packet is dropped at hop 1}] &= \binom{m_e}{m_e} p^{m_e+1} \\
Pr[\text{packet is dropped at hop 2}]^2 &= \binom{m_e+1}{m_e} p^{m_e+1}(1-p) \\
&\ldots \\
Pr[\text{packet is dropped at hop } i] &= \binom{m_e+i-1}{m_e} p^{m_e+1}(1-p)^{i-1} \\
&\ldots
\end{aligned}
$$

The probability that a packet is dropped is the sum of the above probabilities:

$$P_{e2el} = \sum_{i=0}^{i=H-1} \binom{m_e+i}{m_e} p^{m_e+1}(1-p)^i$$

The average number of transmissions O_l for a packet that was dropped is

$$E[O_l] = \sum_{i=0}^{i=H-1} (m_e+1+i) \times \binom{m_e+i}{m_e} p^{m_e+1}(1-p)^i$$

The expected number of packets that are dropped in a k-packet transmission is $k \times P_{e2el}$. Combining this, with the average number of transmissions we can calculate the average energy consumption by packets that were dropped.

$$E[\mathcal{E}_d] = \mathcal{E}_{base} \times k \times P_{e2el} \times \sum_{i=0}^{i=H-1} (m_e+1+i) \times \binom{m_e+i}{m_e} p^{m_e+1}(1-p)^i \tag{5.6}$$

The probability of a successful delivery is

$$P_{e2es} = 1 - P_{e2el}$$

The average number of packets that are successfully delivered are:

$$k_s = k \times P_{e2es} \tag{5.7}$$

[2]Since the last transmission is a failure when a packet is dropped once it runs out of energy budget, we have $\binom{m_e+1}{m_e}$ possibilities for the m_e allowable retransmissions and the one successful transmission over the first hop.

The average number of one-hop transmissions, O_s, that happened for the delivery of a successful packet is[3]

$$E[O_s] = \sum_{i=0}^{m_e} \binom{H+i-1}{i} \times (H+i)(1-p)^H p^i$$

The average energy consumption by successfully delivered packets is given by:

$$E[\mathcal{E}_s] = \mathcal{E}_{base} \times k \times P_{e2es} \times E[O_s] \tag{5.8}$$

Thus, the average energy consumption for the transmission of k packets using JTP when each packet has an energy budget that corresponds to m one-hop transmissions is obtained from Equations (5.6) and (5.8):

$$\begin{aligned} E[\mathcal{E}_{tot}^{JTP}(k,m)] &= E[\mathcal{E}_s] + E[\mathcal{E}_d] \\ &= \mathcal{E}_{base} \times \Big[kP_{e2es} \sum_{i=0}^{m_e} \binom{H+i-1}{i} \times (H+i)(1-p)^H p^i + \\ &\quad + k \times P_{e2el} \sum_{i=0}^{i=H-1} (m_e+1+i) \times \binom{m_e+i}{m_e} p^{m_e+1}(1-p)^i \Big] \end{aligned} \tag{5.9}$$

In JTP we can control the end-to-end loss rate by changing the energy budget of the packets. This means that JTP can control the loss rate and provide specific loss tolerance levels. For any given loss tolerance level and loss probability p, we can assign the energy budget for each packet so as to satisfy

$$\begin{aligned} l_{e2e} &\geq P_{e2el} \Leftrightarrow \\ l_{e2e} &\geq \sum_{i=0}^{i=H-1} \binom{m_e+i}{m_e} p^{m_e+1}(1-p)^i \end{aligned} \tag{5.10}$$

From the above Equation it's hard to get a closed-form solution for accurately computing the energy budget needed to achieve a given loss tolerance; this is further discussed in Section 5.1.5.

[3]Note that the last transmission over the last hop for a successfully delivered packet is always a successful trial.

From Equations (5.7) and (5.9), the energy consumption per bit for JTP is given by:

$$\frac{\partial E[\mathcal{E}_{tot}^{JTP}]}{\partial k_s} = \mathcal{E}_{base} \times \Big[\sum_{i=0}^{m_e} \binom{H+i-1}{i} \times (H+i)(1-p)^H p^i + \\ + \frac{P_{e2el}}{1-P_{e2el}} \sum_{i=0}^{i=H-1} (m_e+1+i) \times \binom{m_e+i}{m_e} p^{m_e+1}(1-p)^i \Big]$$

5.1.5 Numerical Results

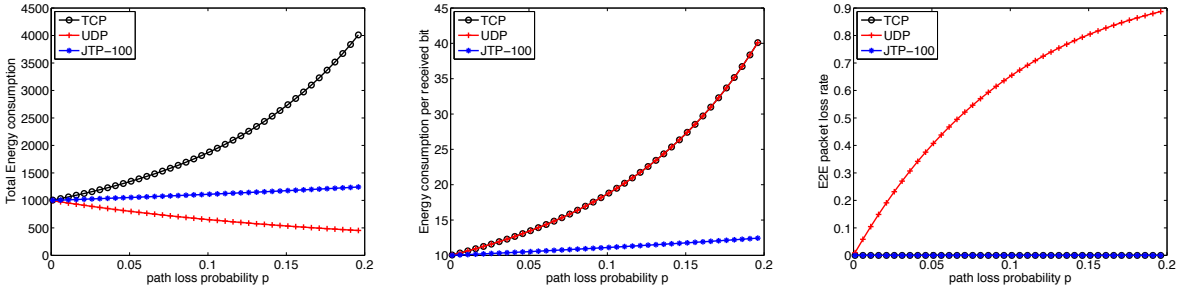
In this section we present numerical results comparing the error control performance of TCP, UDP, and JTP, based on the models described in the previous sections. We used matlab [TM] to numerically solve the model Equations (5.2, (5.1, and (5.9). **JTP-100** refers to JTP configured to provide 100% reliability, i.e. the same reliability level as TCP, i.e. the energy budget assigned to every packet is given by Equation (5.1);

In the first set of experiments, we vary the link loss probability p from 0.001 to 0.2, while keeping fixed the number of hops in the path to 10. In the second set of experiments, we vary the number of hops in the path from 5 to 30 while keeping the link loss probability fixed at 0.05. The number of packets transmitted is kept constant throughout the experiments and equal to 100 packets. The energy $\mathcal{E}_{base}$ for the one-hop transmission is set to 1 unit of energy. Figure 5.1 shows the results.

Although UDP consumes the least amount of energy, it does not deliver all the packets, and thus, has the same energy efficiency as TCP; this is shown by the middle plots in Figure 5.1. JTP can achieve similar reliability as TCP, while consuming much less energy due to the local recovery of packets.

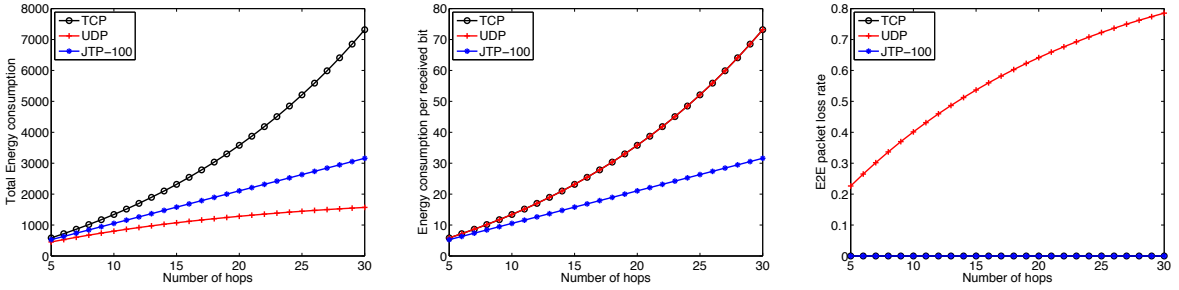
As mentioned previously in Section 5.1.4, if the application does not require 100% reliability it is not straightforward how to set the energy budget in packets so as to satisfy the application's requirements while conserving energy. An idea is to assign to packets a percentage of the extra transmissions, m_e, needed to achieve full reliability.

Figure 5.2(a) shows the end-to-end (E2E) packet loss rate for different percentages of m_e, while we vary the path length between the sender and the receiver from 5 to 50 hops. The link loss probability is fixed to 0.05. JTP-0 is equivalent to UDP since no



Total Energy Consumption Energy per received bit E2E packet loss rate

(a) Comparison of TCP, UDP and JTP with respect to loss probability p

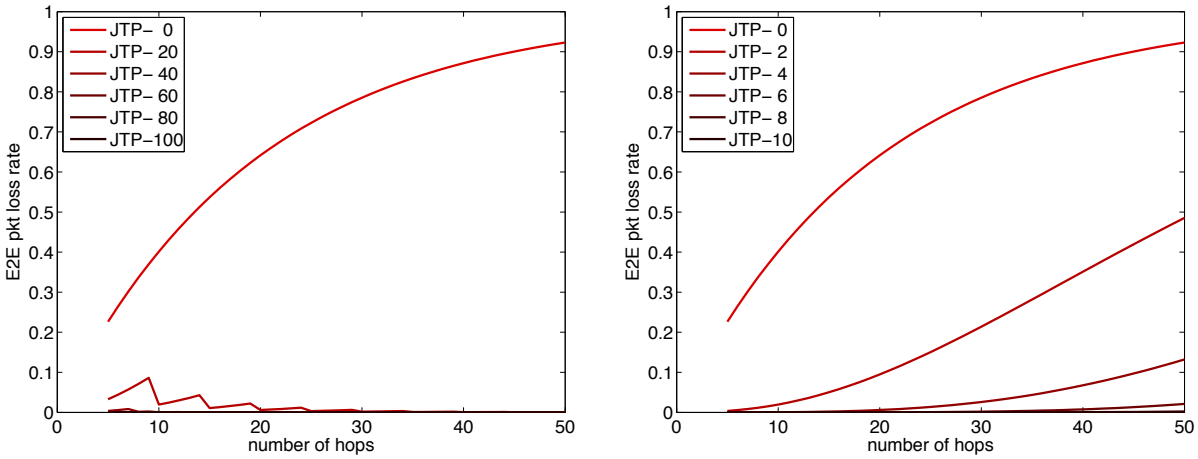


Total Energy Consumption Energy per received bit E2E packet loss rate

(b) Comparison of TCP, UDP and JTP with respect to number of hops in the path

Figure 5.1: Comparison of TCP, UDP, and JTP

extra retransmissions are allowed, and $JTP-100$ is similar to TCP, since it provides 100% reliable transfer of packets. As we can see from the figure, even with 20% of the extra retransmissions the E2E packet loss rate is less than 10%. It's also worth pointing out that unlike the UDP case, as the path gets longer, the E2E packet loss rate decreases. To explain this observation, recall that the number of extra transmissions is computed based on the average number of tries needed to successfully transmit a packet over one hop. Since we are assigning the total number of retransmissions at the source, any unused tries from previous hops are carried along, increasing the number of allowed retries on next hops and thus increasing the success probability. As the path length increases the number of leftover retries that are accumulated also increases causing the E2E packet loss rate to drop. Figure 5.2(b) shows the E2E packet loss rate for different path lengths, when we assign a constant number of extra retries, independent of the path length. As we can see with this approach we can better control the E2E packet loss rate, but as mentioned earlier in Section 5.1.4 it is hard to derive a closed-form solution to compute the exact number of extra retransmissions needed in total at the source. As we describe in the next Section 5.2, to address this problem the number of retransmissions for each hop is determined locally at each node as the packet traverses the path. This not only guarantees a better tuning of the energy consumed but it also allows us to use more accurate information about the link loss rate at every hop. Even with this high level analysis of error recovery mechanisms, it is evident that there is a lot of benefit in recovering lost packets locally at the node where the loss occurred, instead of relying on source retransmissions. JTP attempts to accomplish this goal by using both in-network retransmissions, described in Section 5.2, and in-network caches, discussed in Section 5.3.2, to allow a flow to recover from lost packets as close to the receiver as possible.



(a) Compare JTP using different % of the maximum energy budget

(b) Compare JTP for different number of extra retransmissions

Figure 5.2: Comparison of different reliabilities for JTP

5.2 In-network Retransmissions

A popular approach to counteract the high BER in wireless communications, is to devise techniques that increase the reliability at the link layer, in the hope that making the links appear more reliable to the higher layers, would protect the performance of protocols that are higher up in the network stack. A standard technique used, is to deploy link-layer transmission ACKs and allow for retransmissions of lost frames [PAP+95]. Although this improves the reliability of the link, it also affects other link characteristics, like the transmission delay, which might have undesirable side effects on the performance of other protocols or end-user applications [CLC+08, DMCOC93].

More specifically for a transport protocol that provides reliable data transfers through some type of end-to-end recovery mechanism, the link-layer retransmissions might degrade its performance further due to the inadvertent interaction between the multiple retransmission timeouts [Lud00a]. In TCP for example, multiple retransmissions at the link layer cause high variance in the RTT, leading to spurious timeout, thus degrading its performance. The link-layer is at a loss on how to strike a balance between these tradeoffs since

it does not have all the information about the characteristics of the overlaying traffic. It seems more appropriate that the protocol or application that is generating the packets is able to make these type of decisions since it has all the appropriate data available.

Although in most traditional systems, the ability to affect the number of link layer retries; or any other transmission parameter, is not provided, the JAVeLEN architecture allows protocols that are submitting data for transmission to specify various transmission parameters with the use of transmission profiles; see Section 2.1.2. In the context of this thesis, we explore how the transport protocol can efficiently decide on the number of link-layer retransmissions. The setting of other transmission parameters could also be explored, but it is out of the scope of this thesis. The use of ARQ schemes to improve link reliability is very common in wireless link layer protocols and although the number of retransmissions in most of these protocols is static and can not be modified, it is interesting to explore the benefits of making this parameter dynamic.

5.2.1 Setting Number of Link-layer Transmissions

JTP takes advantage of the opportunity to influence the number of link-layer retransmissions provided by JAVeLEN and makes an informed decision striking a balance between end-to-end reliability, energy consumption and delay, based on the application's requirements.

Given that not all applications require a 100% reliable transfer, JTP can dynamically adjust the number of link-layer retransmissions so as to comply with the application's requirements while reducing the total number of retransmissions. Although the main goal of JTP is to conserve energy, reducing the total number of retransmissions has also a positive effect in conserving bandwidth and reducing the delay.

This mechanism of JTP can be seen as a link-morphing technique that takes a link with basic frame loss rate, transmission delay and bandwidth and adjusts them to better match the needs of each packet while complying with the goal of the network for energy preservation.

Each application expresses its reliability level in terms of a loss tolerance percentage

(e.g. 10% of packets can be lost) which the ATC module encodes in the header of each packet. Based on the loss tolerance and the current frame loss rate experienced on the link to the next hop, JTP adjusts the number of link-layer retransmissions to achieve the desired end-to-end loss tolerance. The MAC can either directly provide the loss rate to JTP, or it can provide per transmission feedback (as done in the JAVeLEN system), enabling JTP to keep its own statistics about the loss rate of JTP packets. iJTP, more specifically the DPS plugin, is responsible for maintaining these statistics.

Let l_{e2e} be the end-to-end loss tolerance requested by the application. Let $n_i,\ i \in [0, H]$ be the nodes on the path from the source, n_0, to the destination n_H, where H is the total number of links in the path. Let $q_i,\ i \in [0, H-1]$ denote the probability a packet sent by node n_i is successfully received by node n_{i+1}.

In order to satisfy the end-to-end loss tolerance requested by the application, the following equation should hold:

$$l_{e2e} \geq 1 - \Pi_{i=0}^{H-1}\ q_i \tag{5.11}$$

The value of q_i depends on the number of link-layer transmission attempts. Let p_i denote the probability that a single transmission from n_i to n_{i+1} fails, i.e. p_i is the packet loss rate. Let M_i denote the total number of link-layer transmission attempts requested for a JTP packet.

In order for a packet transmission to fail, all M_i attempts of the MAC should fail, which means that :

$$q_i = 1 - p_i^{M_i} \tag{5.12}$$

Equation (5.12), tells us that each value of M_i corresponds to a different value for q_i.

There are multiple combinations of M_i values, for $i \in [0, H-1]$, that can satisfy inequality (5.11). Choosing which set of M_i values to use, depends on how much effort each node should put into transferring the packet. Although there are many different strategies that might be employed for assigning an M_i value at each link—e.g. imposing higher number of retries on less loaded links or on nodes with higher available energy—in this thesis we

examine the case in which the M_is are computed so as to achieve equal success probabilities $q_i = q$ for all the links.

If all necessary parameters for computing the different q_is are known at the sender node, the sender could compute the M_i values and encode them in the packet headers. Besides:w the communication cost of exchanging the required information, in MANETs the rate of change for many of these parameters is fast enough that the values could change while the packet is in transit, which will render any values assigned invalid. Consider the case where the path changes while a packet is in transit. New nodes that are now part of the path are at a loss on how to handle the packet since the headers don't carry any instructions for these nodes. JTP deploys a distributed algorithm for computing q_i on each link based only on packet headers and on information available locally in a node. The algorithm is able to auto correct and adjust as the network changes.

Let's consider the case where a packet arrives at node n_j. Node n_j should decide what q_j to use for transmitting the packet to node n_{j+1}. Let l_j be the loss tolerance of the application that corresponds to the path from node n_j to the destination, i.e. the packet should arrive at the destination with probability $(1 - l_j)$. Similar to Equation (5.11):

$$l_j \geq 1 - \Pi_{i=j}^{H-1} \, q_i \tag{5.13}$$

Note that if $l_j = l_{e2e}$ and $j = 0$, this degenerates to Equation (5.11). As we stated earlier, we assume that all q_is are equal, so we rewrite the equation to be:

$$l_j \geq 1 - q^{H_j} \Leftrightarrow q^{H_j} \geq 1 - l_j \Leftrightarrow q \geq (1 - l_j)^{\frac{1}{H_j}} \tag{5.14}$$

where $H_j = H - j$ is the number of links from n_j to the destination. JAVeLEN uses link-state routing, so each node can compute the value of H_j based on its local view of the network graph. Given that we would like to minimize the effort of the network and higher values of q translate to more link retransmissions, we can compute the minimum value of q that will satisfy inequality (5.14), and assign this value to q_j. From Equation (5.12) we

can now compute the value for M_j:

$$M_j = \lceil \frac{\log(1 - q_j)}{\log(p_j)} \rceil \tag{5.15}$$

For applications that request 0% end-to-end loss tolerance, l_j is always 0 and thus q_j is 1 and $M_j = \infty$, which effectively means that the link-layer should keep on retransmitting a packet until it is successfully received by the next node. However, in the dynamic environment of MANETs, many times the transmission fails not just due to increased noise/interference but also due to temporary/permanent link failures caused by mobility or other topological changes. In these cases it is pointless to keep on retransmitting. To avoid pointless retransmissions, the MAC limits the number of allowed retransmissions per packet, to M^{max}, modifying slightly Equation (5.15) to:

$$M_j = \min(\lceil \frac{\log(1 - q_j)}{\log(p_j)} \rceil, M^{max}) \tag{5.16}$$

Given that M_j can only take discrete values, the actual value for q_j will be different from the computed one:

$$q_j^{real} = 1 - p_j^{M_j}$$

Before the node transmits the packet to its next-hop, it must update the loss tolerance field to incorporate the actual success probability that the packet had when being transmitted on this link and to reflect the remaining loss tolerance for this packet from node n_{j+1} to the destination.

$$\Pi_{i=j}^{H-1} q_i = 1 - l_j \Rightarrow q_j^{real} \Pi_{i=j+1}^{H-1} q_i = 1 - l_j \Rightarrow \tag{5.17}$$
$$l_{j+1} = 1 - \frac{1 - l_j}{q_j^{real}}$$

When JTP initially constructs the packet at the source, it assigns $l_0 = l_{e2e}$ to the loss tolerance field. The nodes in the path update that field according to Equation (5.17) ensuring that the probability that the packet will arrive at the destination satisfies application's requirements. By dynamically adjusting the hop-by-hop success probability, JTP is robust

to path changes since if the path is longer or shorter than the sender expects, the values are going to be re-calibrated as the packet goes through the network. Moreover if there is a very reliable link in the path with transmission success probability much higher than the required one by the application, this approach will enable consecutive nodes to reduce their effort.

JTP also provides the flexibility to assign a different loss tolerance for each individual packet, which allows the application to prioritize its packets. For example, in video streaming applications, not all video frames are equally important. The applications can vary the reliability level of each individual ADU satisfying the required quality of service while reducing the use of network resources.

5.2.2 Implementation and Evaluation

The algorithm for computing and setting the number of link-layer retransmissions for each packet needs to be executed by every node in the path, and thus belongs to iJTP. This algorithm has been implemented as part of the DPS plugin. The packet loss rate is computed by the DPS plugin as well, based on the transmission feedback of each packet by the link-layer. iJTP receives an estimation of the path-lengths to all destinations from routing, based on the local view of the network topology. DPS plugin has all the necessary information to perform the computation.

In order to verify the benefits of dynamically adjusting the link-layer retransmissions, we tested JTP under different reliability levels. Three different levels of loss tolerance have been considered: 0% (jtp_0), 10% (jtp_{10}), and 20% (jtp_{20}). In this experiment we compared the different reliability levels of JTP over a linear topology with varying number of hops; x-axis. As we increase the length of the path the end-to-end reliability of the path decreases since there are more chances for a packet to get lost. What we expect to see is that JTP will always provide the requested loss tolerance to the application independent of the path length. For more details about the simulation setup see Chapter 7.

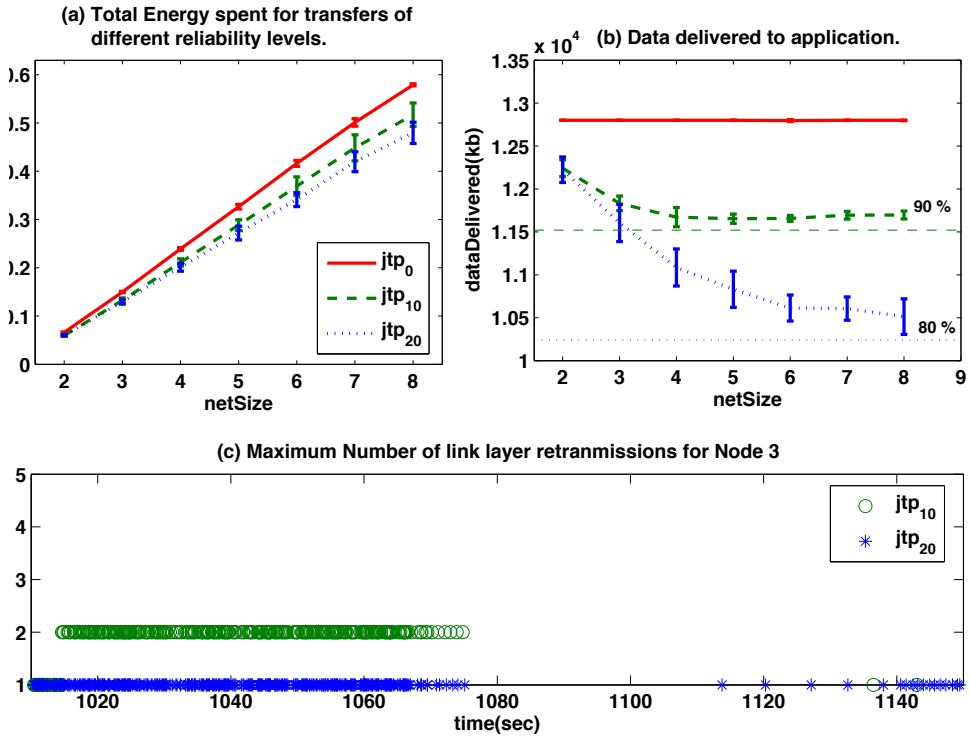


Figure 5.3: JTP performance for different applications with different loss tolerance (0%, 10% and 20%, respectively)

Figures 5.3(a) and (b) demonstrate that by neither over-achieving (e.g. TCP-like full reliability for all), nor under-achieving (e.g. UDP-like no reliability for all), JTP manages to save energy and still satisfy the application's requirement denoted by the horizontal dotted lines in Figure 5.3(b). The jtp_{20} plot is well above the application's requirement since the aggregate loss rate of the path is below 20%. Figure 5.3(c) shows the maximum number of node retransmissions set by iJTP for each packet at the third node along a 4-node path. Packets that request higher reliability are retransmitted more times by the link-layer in cases when the link quality is not good enough. JTP directly affects the amount of network's effort in delivering each packet.The flow that corresponds to 0% loss tolerance is omitted from this plot since iJTP will always assign its packets the maximum number of retransmissions which is 5 for this network.

5.3 End-to-end Retransmissions

Although JTP utilizes link-layer retransmissions to cope with wireless losses, packets might still get lost in the network, due for example to a temporary excessive degradation in link quality that causes packets to be dropped after the maximum number of retransmissions has been reached. Moreover packets might be lost downstream due to buffer overflows or route failures. JTP employs an end-to-end mechanism to recover from lost packets. The destination host requests needed lost packets through Selective Negative ACKnowledgments (SNACKs), that are part of the feedback packets (Section 5.3.1).

In order to prevent packet retransmissions from the source as much as possible, JTP employs in-network caching of packets passing through intermediate nodes. If a packet delivery fails at some later point along the path to the destination, an intermediate node can attempt to re-deliver the packet if it is still present in its cache. In such cases, source retransmissions are avoided, yielding energy savings along the path. The caching and local recovery mechanism is described in Section 5.3.2.

5.3.1 Selective Negative ACKnowledgments

Unlike TCP, JTP employs SNACKs in order to request missing packets. TCP mainly employs cumulative acknowledgments [oSC81]. Cumulative acknowledgments cause the sender to retransmit bytes that have been received out of order, thus wasting energy and unnecessarily using up resources. TCP has been enhanced with a Selective Acknowledgment option [MMFR96] so that the destination can acknowledge out-of-order bytes that have been successfully received. In this case, the source only retransmit all bytes that have not been explicitly acknowledged. Although this enhancement removes the inefficiency of retransmitting the same bytes multiple times, it is still steered toward environments where 100% reliability is required, since the source makes the implicit assumption that all non-acknowledged bytes are needed by the destination.

JTP supports variable levels of reliability and needs to be able to explicitly specify

which packets are needed for retransmission, and thus employs a Negative Acknowledgment [Fox89] scheme, where the destination selectively chooses which packets to request for retransmission. JTP encodes this information in the feedback packets. JTP's scheme, just like in the selective acknowledgment case, employs also cumulative acknowledgments, to signal the sender about data that the receiver will never request in the future.

As shown in Figure 3.9, the feedback packets include the SNACK fingerprint, which is a bitmap of $0s$ and $1s$, for the range between the first packet id and the last packet id. If a specific place in the bitmap is 1, that means that this packet is requested for retransmission.

Representing the requested packets with a bitmap gives us the opportunity to compress it and reduce the total size of the bitmap. Studies that analyze the errors on wireless networks, show that both BER and packet error rate (PER) are not i.i.d. processes but the errors are correlated [LCL07, Jia02, BA03]. They show that there are bursts of packet losses and bursts of successful packet transmissions. In our implementation we use the simple Run Length Encoding (RLE) [Sal07], to compress the bitmap in SNACKs. Although there might be a better compression scheme, RLE is simple and in the worse case the encoding ends up being as big as the original bitmap. A further analysis of encoding schemes that might match better the patterns of wireless losses is out of the scope of this thesis and is left for future work.

5.3.2 Caching

In a MANET environment, packet losses are common even if the network is trying to compensate for the unreliable, shared wireless medium. Even packets that are eventually dropped in the network, might have been successfully transmitted part of the way to the final destination. In a system that strives to conserve energy, every transmission must count, and so even partially successful end-to-end transmissions should be utilized if possible. To that end, JTP employs opportunistic, soft-state, in-network caching, to avoid unnecessary end-to-end retransmissions and enable packets to be recovered by nodes in the middle of the path if necessary. The destination, which has better knowledge of the application's

requirements than mid-path nodes, may request any of these cached packets that it is missing, and are still needed by the application.

Upon receiving a traversing feedback packet, a node checks the SNACK field to determine whether any packet(s) requested for retransmission exists in the node's local cache. Requested packets found in the cache are forwarded downstream toward the destination.

Besides the SNACK field, a feedback packet (Section 3.2.3 also contains in the headers a *locally-recovered packets* field, used to indicate which of the packets requested for retransmission have been already locally retransmitted by some node. Upstream nodes check this field to avoid multiple retransmissions of the same packets. When the source of the transfer receives a feedback packet, it retransmits packets that are present in the SNACK field but are not present in the locally-recovered packets field. The locally-recovered packets field is encoded the same way as the SNACK field.

5.3.3 Cache Size and Replacement Policy

In the analysis above we assumed that the cache has infinite size, i.e. if a packet is received by a node, then there is always space to cache the packet and it is never evicted. So if the destination requests that the packet be retransmitted, the packet will be present in the cache. In real networks however the cache size on the nodes is limited. Even if packets have expiration times, it is still expected that packets will need to be evicted from the cache before they expire. In this section to discuss how JTP addresses this problem.

JTP's goal is to try and make the cache seem infinite from the perspective of a flow. As mentioned earlier, Section 5.1 this effectively means that when a packet is requested for retransmission, if a node has successfully received it then the packet will be present in the node's cache. In other words, the packet should stay in the cache long enough so that if it is lost on subsequent hops, the destination will have a chance to request it.

JTP's series of caches along the path, resembles the Hierarchical Memory Models [AACS87] which was later expanded to Hierarchical Web Caching Systems [CDN+96]. In Hierarchical Web Caching Systems a tree of caches is built, where the caches at the leaf level are accessed

first and if the requested object is not found, the request propagates up the tree, until it reaches the actual web server; the source for the specific object. If an object is found in one of the caches then the object is propagated down the tree until it reaches the client that requested the object. Although in JTP the there is no global hierarchy on caches, each flow builds a hierarchy of caches by storing packets in all the nodes in the path, where nodes closer to the destination are queried first and the request is propagated toward the source through a series of caches. If a packet is found in a node, it is forwarded toward the destination and the packets gets cached in all the nodes on the way.

Che et al in [CWT01] study the behavior of two-level hierarchical caching systems that employ the Least Recently Used (LRU) replacement strategy. The authors identify a characteristic time for each cache which approximates the lifetime of an object in the cache. As they note in the paper, a cache can be viewed as a low–pass filter with a cutoff frequency equal to the inverse of the characteristic time, so for an object to be in a cache, it should have an access frequency that is higher than the cutoff one. In JTP the access frequency of a packet is associated with the frequency of feedback packets, which is discussed in more detail in Section 6.3.

The size of a node's cache is finite and even if packets have expiration times, the cache is still expected to be fully utilized and thus when new packets are received, the node has to make a decision which packets should be kept in the cache and which should be discarded. We broadly classify *cache replacement policies* into time-based, usage-based, and location-based. In the context of JTP, *time-based policies* define the *freshness* of packets in terms of their age in the cache. A simple time-based cache replacement policy is FIFO (First-In-First-Out), in which the packet evicted from the cache is the oldest packet (*i.e.* the one that arrived first). The motivation of FIFO, is to retain newly arriving packets as they may be more likely to be requested soon for retransmission.

Usage-based policies define the freshness of a packet in terms of the elapsed time since the packet was last manipulated, that is, since it was inserted in the cache or since its last retransmission attempt. A simple usage-based cache replacement policy is MRU (Most

Recently Used) or LRU (Least Recently Used), in which the packet evicted from the cache is the most (least) recently manipulated. The motivation behind MRU is to retain those packets that have not been recently requested for retransmission so as to give them a chance to be served from the cache once requested. Alternatively, the motivation behind LRU is to evict those packets not recently requested for retransmission as it may be likely that they will no longer be requested.

Location-based policies would define the freshness of a packet based on the proximity of the packet to its final destination. A simple location-based cache replacement policy is hop-based, in which packets fewer hops away from their destination are given higher priority to be retained so as not to waste the energy expended so far on getting them nearly delivered successfully.

In the context of this thesis we have deployed an LRU strategy. A performance study of different replacement strategies is left as future work.

JTP currently implements FIFO, MRU, and LRU. Since every packet arrival, as well as every negative acknowledgment, that traverse a JTP node, will result in a cache access, the implementation of the caching mechanism has to be efficient in terms of both insertion and lookup times. To this end cache slots are indexed using a hash function, which hashes a *packet signature* to a given cache slot. A packet signature is composed of the 5-tuple (source address, destination address, source port, destination port, packet id). All the packets which are hashed to the same cache slot, are kept in a linked list in an order determined by the cache replacement strategy, e.g. if FIFO is used, then packets are inserted in the end of the list and are removed from the beginning.

5.3.4 Caching Benefits Analysis

In order to compute an upper bound on the gains achieved by caching, we assume a best-case scenario whereby cache sizes are infinite, and the path is symmetric, thus each lost packet will be recovered by the last downstream node which has successfully received it.

We compute the expected total number of node transmissions, denoted by $E[T_{tot}^{JTP}]$. In

the presence of in-network caching, each packet will effectively be retransmitted over each link for as many times as needed, until it is successfully delivered to the next node. For the purpose of this analysis the SNACK transmissions are not considered. The purpose of this analysis is to study the effect of caching in packet energy savings, the SNACK packets have to be transmitted to signal loss independent of where the packet is going to be recovered from. If p is the packet loss probability of the link, the expected number of node transmissions on a link l follows a geometric distribution with mean $E[T_l^{JTP}] = \frac{1}{1-p}$.

The expected total number of node transmissions required by JTP in order to deliver k packets over H hops is given by:

$$E[T_{tot}^{JTP}] = k \times H \times \frac{1}{1-p} \tag{5.18}$$

JTP without Caching

In the case of JTP with no in-network caching (henceforth denoted by JNC), over each link, the packet is transmitted, say, at most M times. If its transmission still fails, then it must be retransmitted from the source.

Denote by $S > k$, the random variable representing the total number of packets sent by the source until k packets are successfully delivered at the destination, then $E[S] = \frac{k}{q_{e2e}}$, where q_{e2e} is the end-to-end success probability.

When a packet is received at a node, the average number of node transmissions that it

triggers on link l is:

$$\begin{aligned}
E[T_l^{JNC}] &= (1-p) + 2(1-p)p + \cdots + M(1-p)p^{M-1} + Mp^M \\
&= \frac{1-p}{p} \times \sum_{i=1}^{M} ip^i + Mp^M, \text{ known power sum} \\
&= \frac{1-p}{p} \times \frac{p}{(1-p)^2} \times \{1 - p^M[1 + M(1-p)]\} + Mp^M \\
&= \frac{1 - p^M[1 + M(1-p)]}{1-p} + Mp^M \\
&= \frac{1 - p^M - M(1-p)p^M + (1-p)Mp^M}{1-p} \\
&= \frac{1-p^M}{1-p}
\end{aligned}$$

Given that the link success probability is $q = (1-p^M)$, the probability that a packet makes it over i links is q^i, where each packet then triggers $E[T_l^{JNC}]$ node transmissions. $q_{e2e} = (1-p^M)^H$,

Recall here that $E[S] = k/q_{e2e}$ and note that $q_{e2e} = (1-p^M)^H$. Thus, the total number of node transmissions for JNC is given by:

$$\begin{aligned}
E[T_{tot}^{JNC}] &= \sum_{i=0}^{H-1} E[S] \times q^i \times E[T_l^{JNC}] \\
&= M \times \frac{1-p^n}{1-p} \times \sum_{i=1}^{N} P_s^(i-1) \\
&= M \times \frac{1-p^n}{1-p} \times \sum_{i=0}^{N-1} P_s^i \\
&= \frac{M(1-p^n)}{1-p} \times \frac{1-P_s^N}{1-P_s} \\
&= \frac{M(1-p^n)}{1-p} \times \frac{1-P_s^N}{1-P_s} \\
&= \frac{k(1-p^n)(1-(1-p^n)^H)}{(1-p^n)^H(1-p)p^n} \approx \frac{k \times H}{(1-p^n)^{H-1}(1-p)}
\end{aligned} \tag{5.19}$$

For $H = 1$, Equation (5.19) degenerates to (5.18). Observe that the cost of JNC is $\frac{1}{(1-p^n)^{H-1}}$ times higher than that of JTP.

Figure 5.4 shows the per-node energy savings in a 7-hop linear topology. These results are from OPNET simulations, see Chapter 7 for simulation details. As seen in the figure, caching not only reduces the energy consumption at each node, but it also allocates the energy spent for a flow more fairly between nodes, since nodes that are in the beginning of the path do not have to constantly keep retransmitting the same packets as happens when packets losses are recovered from the source.

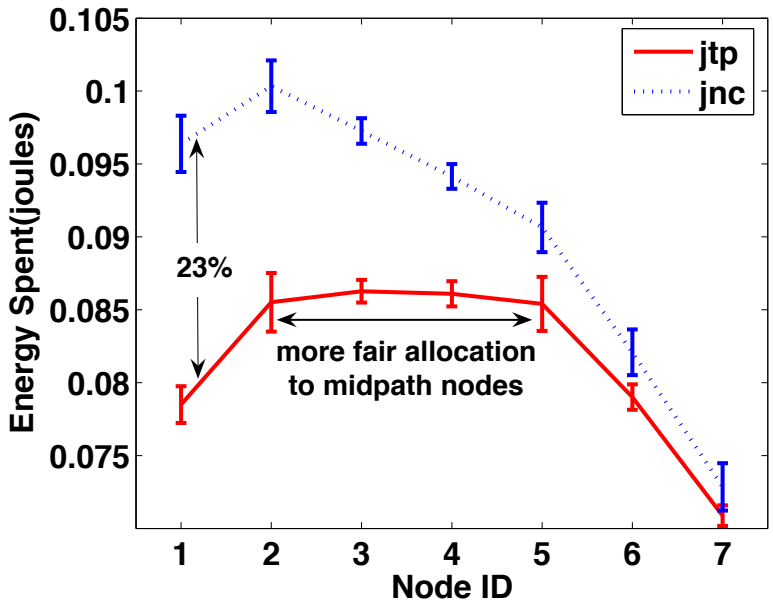


Figure 5.4: Per-node energy consumption for JTP and JTP without caching.

5.3.5 Route Symmetry

In order for caching to yield maximum energy savings, communication paths must be symmetric. Recall that JAVeLEN employs next-hop routing to forward packets using increasingly more accurate topological views toward destinations. Next-hop routing also has lower overhead compared to source routing approaches. Thus, one way to force routes to be symmetric is to ensure that link metrics are symmetric. Recall that the link metric in JAVeLEN

is a quantized function of signal-to-noise ratio. To increase metric symmetry, JTP employs a contention-avoidance flow rate control mechanism, thus it minimizes contention perceived at both nodes of a link. Together with quantization, this maximizes the chances of symmetric link costs and thus of symmetric routes. Figure 5.5 shows the number of cache hits and source retransmissions for 5 JTP flows over a 25-node random topology. (See Chapter 7 for simulation details.) The symmetry of paths is demonstrated by the in-network cache hits experienced by flows, resulting in more than two-third of the lost packets recovered from caches rather than from sources.

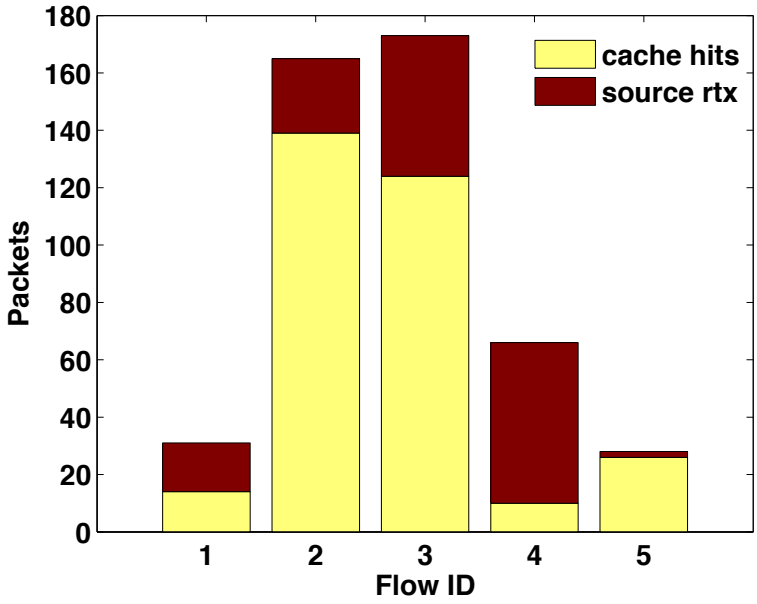


Figure 5.5: Relation between end-to-end and locally recovered packets, in a random topology.

5.4 Conclusion

This chapter discussed the error-recovery mechanisms of JTP and how they affect the total energy spent in a connection. JTP uses dynamic in-network retransmissions to adjust the effort the network is placing in transmitting one packet. The number of retransmissions

are adjusted based on the reliability requirements of the application and on current loss experienced on a link.

JTP also deploys a series of soft-state caches in order to recover losses as close to the destination as possible. For the purpose of this thesis we used a multilevel caching mechanism that caches a copy of the data on every node and uses LRU as the eviction policy. Researchers have recently started looking into the problem of improving the performance of cooperative multilevel caching, especially in the environment of networks. There is work that looks into how to help a node decide whether information should be cached or not locally in order to improve the overall performance and enhance the use of each local cache [LCS06, PCP12]. There is also work in analyzing multilevel caches [RKT10, LCF$^+$10] in order to better understand their performance and compare different methods. As future work it would be very interesting to study the applicability of such approaches within JTP.

In this chapter we provided a high-level analysis for both link-level error recovery and cache management in order to study their effect on energy consumption. We have also studied them in simulations and presented results that prove their benefits.

Chapter 6

Destination-based Transfer Control

It is important for a transport protocol that is envisioned to support reliable or semi-reliable data transfers to incorporate mechanisms for controlling the sending rate as to not overwhelm the receiver, nor the network. These mechanisms include a *flow control* and a *congestion control* mechanism. For this thesis we adopt the definitions as described by Jain in [Jai86]. In summary, flow control ensures that the source and destination are in sync; i.e. that the receiver does not drop received packets because its buffer is full. Congestion control aims at constraining the speed at which a source is injects data into the network so that it does not overwhelm the network, causing packets to be dropped at switches.

Both of these mechanisms of transport protocols become even more crucial in an energy conserving environment where each dropped packet translates to wasted energy. Both flow and congestion control affect how fast the source puts data into the network, i.e. are used to set the *sending rate* of the transfer.

Flow control, [CK74] is needed to manage the data transfer between hosts in order to prevent a fast source from over-running a slow destination node. The buffer capacity at the destination, although not necessarily constant, is slowly and predictably changing, and the destination can communicate this information, commonly called the receiver window, back to the source. Although computing the receiver window so that there are no packet losses at the receiver is tricky, [oSC81] provides a set of good practices about how to set

this value.

Congestion control is necessary so that the source paces itself based on the network conditions to control (or avoid) congestion in the network. Although congestion control was originally implemented to protect the network from collapsing [Jac88], it is also very important for transport protocols since it reduces the packets drops within the network. Unlike the receiver's buffer's capacity, network's capacity changes unpredictably, and it needs constant monitoring. This is especially true in wireless networks where the capacity of a path is not only affected by other flows, but also by transient network conditions; node density, current SNR values etc.

JTP's goal is to avoid congestion in the network instead of controlling it, so as to eliminate if possible packet losses due to overloading the network. In order to achieve this, JTP monitors the network closely and quickly adapts to efficiently avoid congestion, while providing good performance to the applications. There are tradeoffs as to how quickly it is useful for JTP to adapt to changes, since adapting too fast might lead to large oscillations that will degrade overall performance [HDPT04]. Looking at TCP's design, we see that TCP infers the condition of the path by the rate of feedback (ACK) packets and by losses inferred from information carried within these packets. This mechanism assumes that the forward, i.e. the path that the data packets are traversing, and the backward path, i.e. the path that the ACK packets are traversing, of a flow are symmetric. This assumption generally holds in wired but not in wireless networks. Moreover in TCP the rate control is directly dependent on the feedback packets. As described [APS99], the ACK "clock" enforces the rate by which new data is transmitted. Feedback packets are not only pure overhead in a data transfer, but in wireless networks they compete for resources with the data packets due to the shared nature of the communication medium. So although TCP's design is appropriate for wired networks, it is not fitted to face the challenges of wireless networks.

Instead of using feedback packets to infer the condition of the path, JTP attempts to directly monitor the forward path by collecting information from each hop that the packet

traverses. Monitoring data is piggy-bagged in JTP data packets, enabling the destination to observe the status of the forward path. In JTP flows, the destination is responsible for determining the sending rate, which is communicated back to the source in feedback packets. JTP is able to monitor the actual path that the data is traversing and also decouples the path monitoring from the feedback packets. This decoupling enables JTP to separately control the rate of feedback packets, reducing overall network traffic.

6.1 Sending Rate Control

JTP is a rate based protocol, where the sending rate for the transfer is set by the destination. The destination collects the value for the available capacity on the forward path, based on the information included in data packets. JTP employs a DPS-type [SZ99] mechanism, where each node includes in the packet the available capacity to itself from the previous hop on the path. When it is time to send a feedback packet, the destination uses the value of the available capacity in order to set the new sending rate of the transfer.

This system for setting the sending rate is a typical closed-loop feedback system; see a simplified diagram in Figure 6.1. There has been significant work in the computer networking literature that discusses control theoretic approaches to congestion control [BM93] and especially in work that was done for setting the sending rate in ATM networks [KR97, MCG96, KR99]. [YR95] provides a very good taxonomy of congestion control algorithms in computer networks. JTP employs a closed-loop, explicit feedback congestion avoidance controller.

PI^2/MD Data Rate Controller

In JTP we use a closed-loop controller since they are quicker and more efficient in fine tuning a system [FPEN05]. Moreover due to the challenges of wireless networks [BPSK97, SAHS03] an explicit feedback approach is more appropriate since it gives exact and timely feedback for the condition of the network.

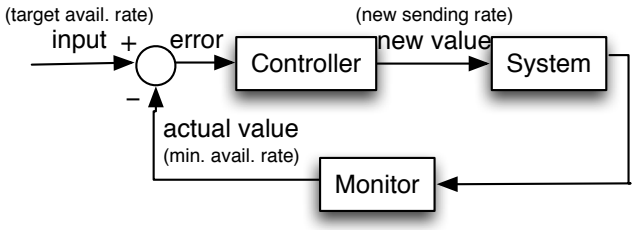


Figure 6.1: JTP's closed-loop Rate Control System.

In JTP, we designed a *PI* style controller [FPEN05] to adjust the sending rate of a flow that is appropriate to the dynamics of a wireless network. A simple proportional controller would adjust the current sending rate only based on the latest measurement, but this would lead to steady-state errors [FPEN05]. Adding an integral part to the controller ensures that the controlled value converges in steady state, and that the error also converges to 0. Although JTP does not include a derivative component in its controller, it does monitor the rate of change of the error; see Section 6.2, so as to react to long lasting changes in the network while not adjusting to short-term variations.

In computer networks, unlike classical control systems, the controller of each of the flows that are sharing a bottleneck link is monitoring and controlling the same values, forming a distributed control system [YR95]. To address this challenge and to ensure that the flows get a fair allocation of the capacity, JTP combines different approaches to build an efficient rate controller. In this thesis when we refer to fair allocation between flows we refer to the max-min fairness [BG92].

JTP deploys a PI^2/MD (Proportional Increase/ Multiplicative Decrease) controller for adjusting the data rate that not only is stable in steady state, it also converges to a fair allocation between flows.

The PI^2/MD controller uses the minimum available rate measured along the path of a JTP connection to control the sending rate of the connection. The available rate of a node represents its current available reception capacity; see Section 6.4.1 for more details. Although, other node-level information such as queuing delay or energy expended per suc-

cessfully delivered bit would be helpful if available, in this thesis we concentrate on the available rate metric that is measured by JAVeLEN's MAC.

$\hat{A}$ denotes the minimum available rate measured at the JTP destination. The available rate value must be estimated over a *long timescale* to avoid unnecessarily reacting to *transient* changes, a common problem in some transport protocols including TCP. We achieve this by using an integral-based controller that uses Exponential Weighted Moving Average (EWMA) for computing the monitored value. If $\hat{A} > \delta$ then the source increases its sending rate r in proportion to the current available capacity and, to improve fairness among competing flows, inversely proportional to the current sending rate.

$$r(t+1) \quad = \quad r(t) + K_I \frac{\hat{A}(t)}{r(t)},\ 0 < K_I < 1 \tag{6.1}$$

On the other hand, if the available rate approaches 0 ($\hat{A} < \delta$), the source decreases its sending rate multiplicatively. This is akin to the multiplicative decrease of TCP that is shown to converge to a fair allocation when there are multiple competing flows [CJ89]. At JTP we decided to back off when the available rate is close to 0 but not 0, because there should always be a small portion of the capacity available for new flows to get started without causing congestion and unnecessary packet drops.

$$r(t+1) \quad = \quad K_D r(t),\ 0 < K_D < 1 \tag{6.2}$$

The average channel utilization is computed as follows:

$$\hat{x}(t) \quad = \quad (1-\alpha) \times \hat{x}(t-1) + \alpha \times \sum_{i=1}^{N} x_i(t)$$

where N is the total number of flows competing over the channel, and $x_i(t)$ is the rate of flow i at time t.

Figure 6.2 shows the instantaneous rates of two JTP sources (i.e. $N = 2$) competing over a channel of a fixed capacity $C = 40$ by numerically solving Equations 6.1 and 6.2 In

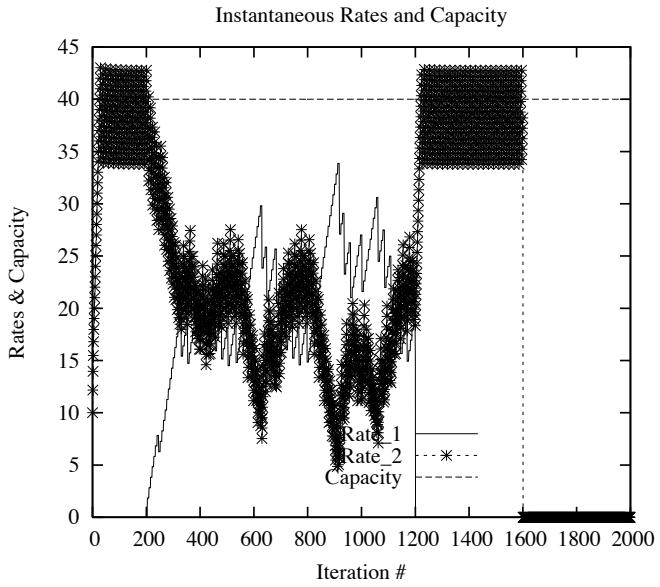


Figure 6.2: Two JTP sources competing over a fixed-capacity channel.

this example, $K_I = K_D = 0.8$ and $\alpha = 0.3$. Flow 1 starts at time 200 at a sending rate of zero, and terminates at time 1200. In addition, flow 1 chooses to update its sending rate only every 6 time units. Flow 2 starts at time zero at an initial sending rate of 10, and terminates at time 1600. We observe that regardless of this heterogeneity in update times, initial sending rates, and start/end times, JTP sources adapt to their fair share as well as make full use of the available capacity.

6.1.1 Stability Analysis of PI^2/MD Rate Controller

In order to analyze the stability of the controller, we consider a single JTP flow adapting its sending rate over a fixed-capacity channel. We use Lyapunov functions to analyze stability of this non-linear system, which is a standard analytical tool in control theory for examining the stability of a controller [FPEN05]. We show that $K_I > 0$ and $K_D < 1$ are sufficient conditions for convergence. We also prove that the controller is efficient and it will converge even for low frequency of sending-rate update albeit at a slower pace.

Consider a single JTP flow adapting its sending rate over a fixed-capacity channel. For analytical tractability, let's ignore the EWMA computation of the available rate, that is, if

$r(t) < C$, then the JTP source adapts its rate as follows:

$$r(t+1) \quad = \quad r(t) + K_I \times \frac{(C - r(t))}{r(t)} \tag{6.3}$$

On the other hand, if $r(t) > C$, then the JTP source adapts its rate as follows:

$$r(t+1) \quad = \quad K_D \times r(t) \tag{6.4}$$

Observe that the system remains non-linear, with two operating regions determined by whether the sending rate $r(t)$ is less than or greater than the capacity C. We next consider each of these two regions, and prove stability by showing that the value of a positive Lyapunov function $V(r)$ decreases with each iteration.

• $r(t) < C$ *Region:* Define $V(r) = C - r$. Then:

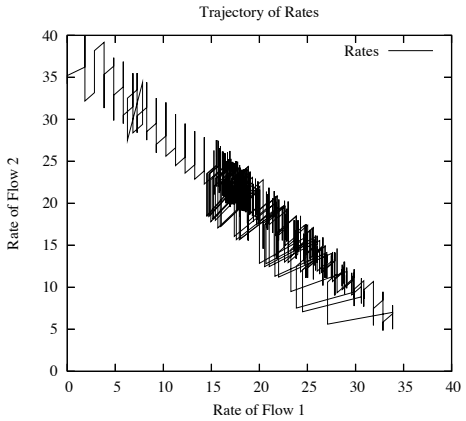
$$\begin{aligned} V(r(t+1)) - V(r(t)) &= (C - r(t+1)) - (C - r(t)) \\ &= C - (r(t) + K_I \times \frac{C - r(t)}{r(t)}) - (C - r(t)) \\ &= -K_I \times \frac{C - r(t)}{r(t)} \\ &< 0 \end{aligned}$$

Thus, the only condition for $V(r)$ to decrease is that $K_I > 0$, regardless of the exact value of K_I. Of course, the exact value of K_I determines the tradeoff between speed of convergence and quality of the steady-state behavior—a higher value of K_I leads to faster convergence but higher oscillations.

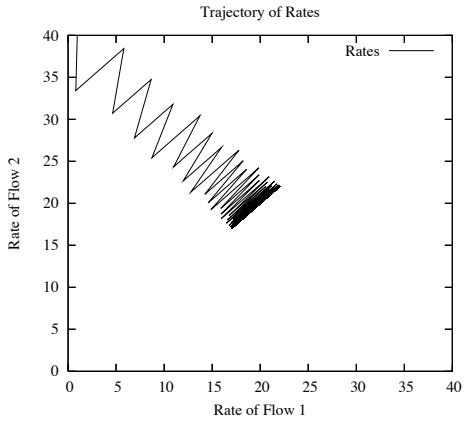
• $r(t) > C$ *Region:* Define $V(r) = r - C$. Then:

$$\begin{aligned} V(r(t+1)) - V(r(t)) &= (r(t+1) - C) - (r(t) - C) \\ &= K_D \times r(t) - C - r(t) + C \\ &= -r(t) \times (1 - K_D) \\ &< 0 \end{aligned}$$

Observe that, for $V(r)$ to decrease, it is required that $K_D < 1$.



(a)Two JTP sources converging to fairness over a fixed-capacity channel. Flow 1 updates its sending rate less frequently, once every 6 time units.



(b)Two JTP sources converging to fairness over a fixed-capacity channel. Both flows update their sending rate at every time step.

Figure 6.3: Convergence of two competing flows.

Thus, $K_I > 0$ and $K_D < 1$ are sufficient conditions for convergence. Furthermore, at steady-state as $t \to \infty$, substituting $r(t+1) = r(t)$ in Equation (6.3), we have $r(t) \to C$, hence the rate control is efficient.

Observe that in the case of lower frequency of sending-rate update, the above analysis still applies, *i.e.* the system converges albeit at a slower pace.

6.1.2 Convergence to Fairness

In order to see whether the rate controller convergences to fairness, we consider two flows adapting their rates according to Equations 6.3 and 6.4. Figure 6.3(a) shows the trajectory of flow rates over the time period [200:1200], during which flow 1 is present. Flow 2 is a long-standing flow that is stabilized when Flow 1 starts. The initial sending rate for each flow is 1 packet per second, and the capacity of the channel is 40 packets per second. Flow 1 updates its sending rate less frequently, once every 6 time units. Both flows eventually converge to their fair share, although the infrequent rate update by flow 1 slows down this convergence and initially causes wide oscillations.

However, if both flows update their sending rates at the same higher frequency, Fig-

ure 6.3(b) shows that convergence is faster and less oscillatory, as expected.

6.1.3 Accounting for In-network Caching

JTP employs in-network caches in order to recover from lost packets as close to the destination as possible; see Section 5.3.2 for more details. Enabling mid-path nodes to retransmit packets on behalf of sources may cause a temporary increase of the actual packet rate of the flow that is greater than the one set by the destination.

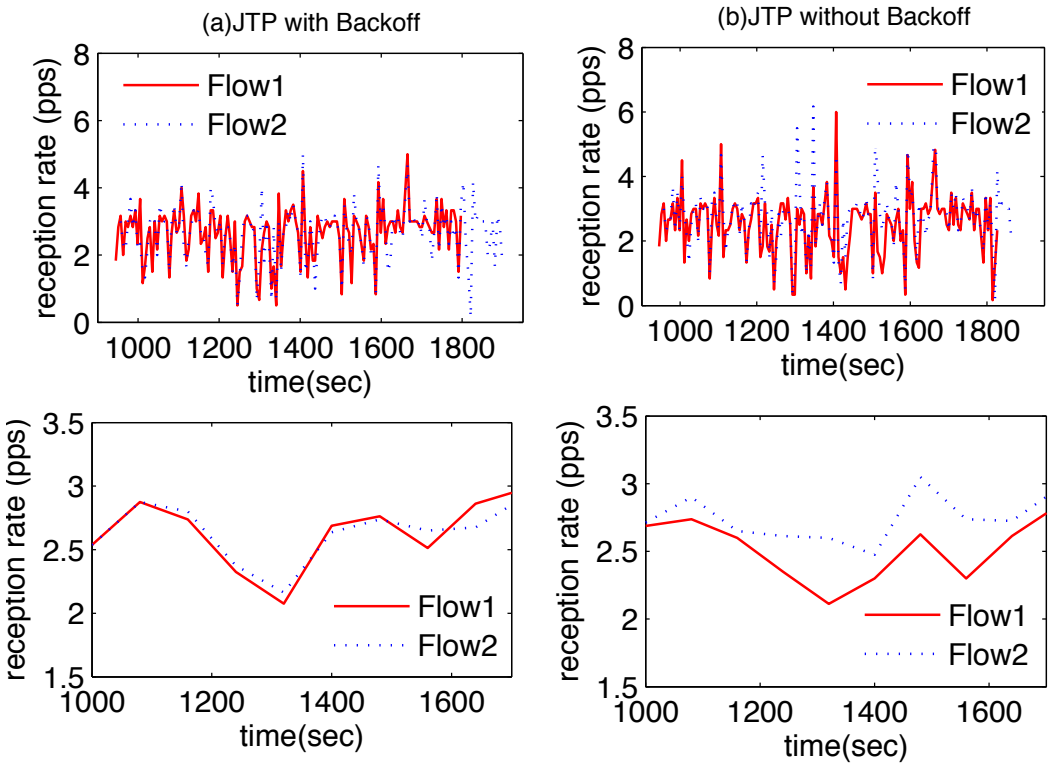


Figure 6.4: Short-term (top row) and long-term (bottom row) average of the reception rate for two competing flows: (a) the source backs off for locally recovered packets; (b) it does not.

To the best of our knowledge JTP is the first protocol that tries to compensate for in-network transmissions in order to achieve a fair allocation of the capacity. In order for flows to fairly share the network, independent of their reliability requirements, and in order to effectively avoid congestion, the source node must incorporate in-network retransmissions

in its sending rate calculations. To this end, JTP sources back off in accordance to the extra traffic that is induced by in-network retransmissions. When a feedback packet is received, the source uses the *locally-recovered packets* field (see Section 3.2.3) to adjust its sending rate. Let $r(t)$ be the rate indicated to the source by a received feedback packet at time t. Let N be the number of packets recovered within the network, and let $s_j,\ j \in [1, N]$ be the sizes of these packets. The source computes the appropriate back-off period t_b as follows:

$$t_b = \frac{\sum_{j=1}^{N} s_j}{r(t)}$$

Figure 6.4 shows the short- (top plots) and long- (bottom plots) term average of the packet reception rate (throughput) at the destination for two competing flows. This results are for a linear topology, with two competing flows. Section 7.2 contains all the details for simulation setup for experiments using a linear topology. Flow 1 does not request packet retransmissions (i.e. UDP-like flow), while flow 2 requires that all its packets be delivered and thus invoking the local recovery mechanism of the in-network caches. We observe in the right plots–plots corresponding to throughput when the sender does not back off after local retransmissions– that there are spikes in the reception rate of flow 2. This is because when the sender does not back off its sending rate to account for its additional in-network retransmissions the flow effectively increases temporarily its fair share above its fair share. The introduced unfairness is more evident from the long-term average plots.

6.2 Path Monitoring using Flip-flop Filtering

In order to effectively react to network changes, JTP needs to constantly monitor the data path and adapt the transmission parameters accordingly. As mentioned earlier the path monitoring mechanism of JTP is placed at the destination. The destination collects various path statistics and monitors their values. In wireless networks transient changes are frequent and JTP needs to strike a balance between fast adaptation to network changes – path changes and available capacity variations – and unnecessary reaction to transient events – temporary increase of loss rate and path re-computations.

If the monitor is aggressive in adapting to the recent values, it might unnecessarily react to temporary, random events and thus constantly adapting, leading to inefficient use of the resources. On the other hand, if the monitor slowly adapts to network changes, it will be sluggish in reacting to more permanent changes, like modifications in the path length due to topology changes, which can lead to congestion and undesired packet losses.

To this end, JTP employs a flip-flop filter [BM02], that although it filters out temporary fluctuations in the monitored values, it is able to quickly adapt to more long-lasting changes. The flip-flop filter has two modes of operation:

- **Stable mode:** The filter is in stable mode, when the conditions in the path are steady, and there are no significant changes.
- **Agile mode:** The filter is in agile mode during persistent changes in the network, and until the monitored values stabilize again.

Let x_i denote the i^{th} sample of a path's metric, $\bar{x}$ the estimated average, and $\bar{R}$ the estimated range of sample values. We use principles from statistical quality control [Mon05] to detect a *significant* change in the path's state, which then triggers a significant-change event.

We estimate the EWMA's $\bar{x}$ and range $\bar{R}$ as follows:

$$\begin{aligned} \bar{x} &= (1-\alpha)\bar{x} + \alpha x_i, \text{ initially } \bar{x} = x_0 \\ \bar{R} &= (1-\beta)\bar{R} + \beta \mid x_i - x_{i-1} \mid, \text{ initially } \bar{R} = \frac{x_0}{2} \end{aligned} \tag{6.5}$$

$\bar{R}$ is used to estimate the deviation around $\bar{x}$. $\bar{R}$ is calculated *only* from samples x_i within the following upper and lower control limits:

$$UCL = \bar{x} + 3\frac{\bar{R}}{1.128}; \quad LCL = \bar{x} - 3\frac{\bar{R}}{1.128} \tag{6.6}$$

Under normal operation, *stable* EWMA filters are employed, i.e. the weights α and β are small so short-term variations are filtered out. As long as x_i lies within the control limits, the state of the connection's path is considered *stable*. If x_i is outside the control limits it

is considered an *outlier*. A certain number of *consecutive* outliers is used as indication of significant *and* persistent change in the state of the path. At this point, the monitor at the destination switches to an *agile* EWMA filter where larger α and β values are used, so that $\bar{x}$ quickly catches up with the actual value. Once x_i falls back again within the control limits, the controller switches back to the stable mode.

Flip-flop filters can be used to monitor and estimate the value for any transmission parameter. Currently in JTP we use them to monitor the available rate of the path as well as the one-way-delay.

6.3 Variable-rate Feedback

Although feedback packets carry valuable information, they are usually pure overhead in a data transfer, since they do not carry application data in a typical unidirectional transfer. Moreover, in wireless networks, due to the shared nature of the communication medium the data and the feedback stream compete for resources and thus frequent feedback packets decrease the performance of the data transfer.

In order to address this problem, the delayed ACK mechanism [Bar89] of TCP is used. Although enabling delayed ACKs, improves the performance [AJ03] of TCP in MANETs, it is hard to decide how often to send an ACK packet [CLGS06]. Researchers have proposed various techniques for dynamically adapting the parameter d in the delayed ACK mechanism [AJ03, CLGS06, OOBB04], where d defines after how many received data packets will an ACK be sent by the receiver, for example if d is set to 2 then the receiver will acknowledge every other data packet. Although these approaches do reduce the feedback traffic and improve TCP performance they are still tied to TCP and its self-clocking and thus have to ensure that enough ACKs are sent for TCP to operate efficiently.

In JTP we deploy a variable-rate feedback mechanism that while reducing the total number of feedback packets, it is able to quickly react to network changes faster than a constant-rate feedback mechanism. JTP sends periodic feedback in a very low frequency,

and sends additional feedback packets only when necessary.

The receiver sets a periodic timer that every time it expires, it sends a feedback packet to the sender with the latest information on the transfer. Except for waiting for the feedback timer, the receiver might send a feedback packet earlier if necessary. Specific reasons for sending a feedback packet sooner, include:

- **significant change in the network's monitored values.** When the path monitoring module changes from stable to agile it signifies an important change in the network. The receiver recomputes the transfer parameters and when necessary sends a feedback packet sooner, and resets the timer.

- **the Application Transfer Controller initiated a new feedback packet.** ATC is responsible for satisfying the requirements of the applications, and monitors the receiver buffer as well as packets that should be requested for retransmission. Each ATC might have different reasons for requesting an early feedback, for example an ATC that supports real-time applications might request an early feedback packet to be sent in order to request the recovery of lost packets before their deadline expires.

- **a probe packet is received.** The sender might request for a feedback packet to be sent. This is usually the case when the sender has not received any feedback packet within the expected time period. The receiver includes in the feedback packet the expected duration between feedback packets indicating to the sender when the next feedback packet should be expected. This is necessary so that the sender has a mechanism of determining whether the receiver is still active and to be able to recover from lost feedback packets. If the sender does not receive a feedback packet by the expected time, it sends a probe packet, requesting that a new feedback packet is sent.

Setting the Feedback Period

When deciding what should be the frequency at which the receiver should send feedback packets there are multiple factors to consider. In our implementation, we set the feedback

period T, as a function of the sending rate and of the cache size used in the network. Specifically,

$T = \min(T_{Lower\_Bound}, n \times \frac{1}{SendingRate}); n \geq 1.$

Notice that this ensures that the destination does not send feedback packets at a rate higher than the sending rate, i.e. the maximum rate at which it receives packets from the network and thus new monitoring information. The value of T_{Lower_Bound} is dependent on the size of in-network caches, since if packets requested for retransmission by a feedback message have already been evicted from the cache then the energy savings achieved by infrequent feedback messages would be offset by the energy consumed by packets that have to be retransmitted from the source. If C is the cache size and RTT is the round-trip time for the connection then:

$$T_{Lower\_Bound} \leq \frac{C}{SendingRate} - RTT$$

The cache at each node is shared between all flows and estimating the actual cache size that is available for a flow is hard. It is best to set the value for the parameter C conservatively.

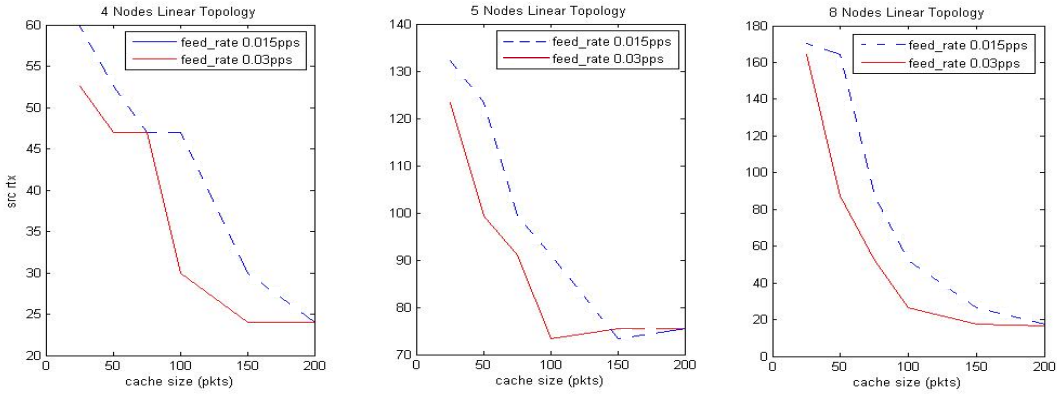
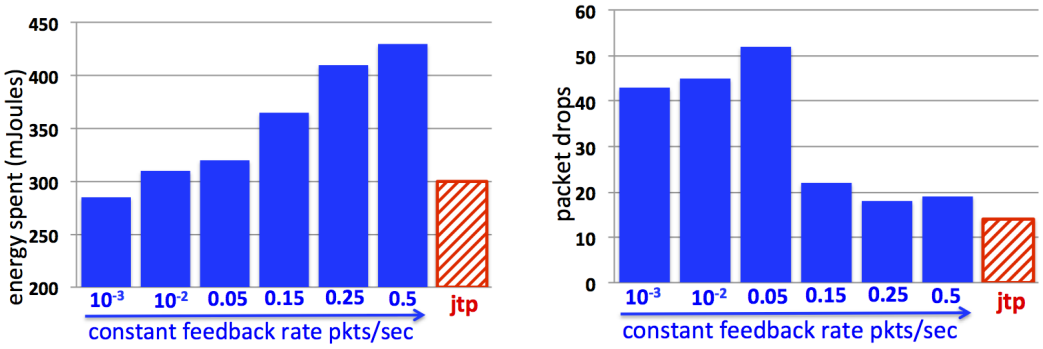


Figure 6.5: The effect of cache size for various network sizes.

Figure 6.5 shows the effect of cache size on the performance of JTP for different network sizes and feedback rates. In this experiment we considered linear networks of different sizes. The reason why we increased the path length between the sender and the receiver is to study



(a) Total energy expended in the system.

(b) Total number of packet drops in the queues of the system.

Figure 6.6: The gains achieved by using variable-rate feedback.

the effect of different RTT values on the effectiveness of the cache. Chapter 7 provides all the details about how the linear topology simulations are setup. In this experiment there is one JTP flow traversing linear networks of various sizes. The figure depicts the number of source retransmissions for increasing cache sizes. We observe the sudden drop in the number of source retransmissions once the cache size is large enough to hold missing packets until they are requested for retransmission from the caches. Increasing the cache size further does not improve the performance significantly.

6.3.1 Comparison with Traditional Constant-rate Feedback

In order to evaluate the performance of the variable-rate feedback mechanism, we compared it against constant-rate feedback implementations. We have used different values for the constant-rate feedback and we discovered that our variable-rate feedback mechanism has more timely reactions to network changes.

Figure 6.6 depicts the energy gains achieved by using variable-rate feedback instead of a constant rate. In this experiment, a linear topology of 8 nodes is used, with one long-lived flow competing with several short-lived ones. (See Section 7.2 for simulation details.) The short-lived flows provide an unpredictable available capacity in the network that enables us to study how well the variable rate feedback mechanism performs under a highly varying

environment.

For low values of the feedback rate, we observe a high number of packet drops in the queues of intermediate nodes (Figure 6.6(b)). This is caused by the slow reaction of the long-lived flow to the changing network conditions. The source does not back off fast enough causing congestion in the system. As we increase the rate of the constant feedback, the total energy consumed (Figure 6.6(a)) increases since more feedback packets are generated consuming network resources. Using variable-rate feedback, JTP not only achieves low energy consumption, but also significantly reduces packet drops in the system since whenever the system load increases, the receiver sends a timely feedback forcing the source to back off.

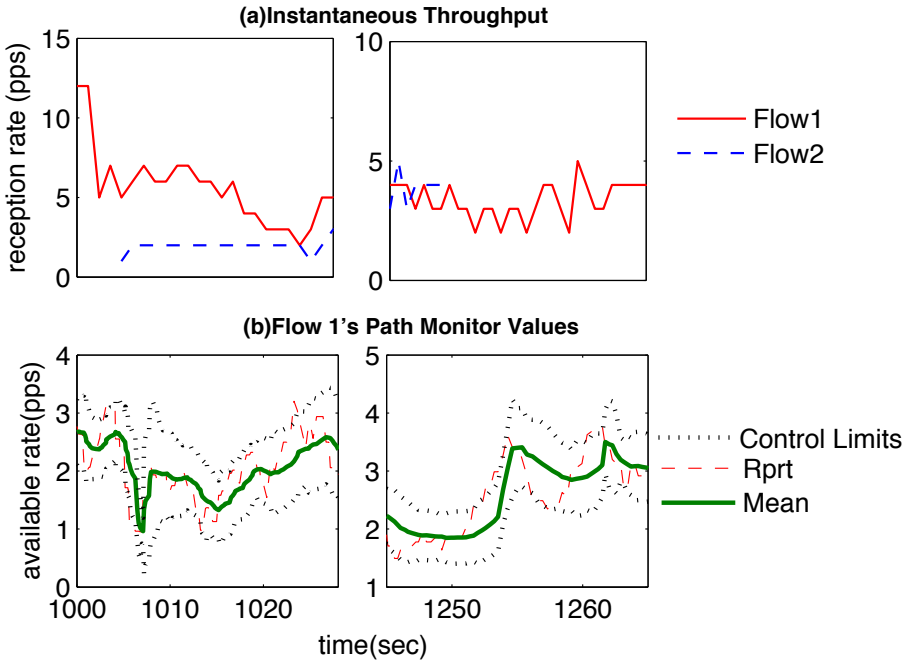


Figure 6.7: Rate adaptation for two competing JTP flows.

Figure 6.7 demonstrates the behavior of the variable-rate feedback mechanism when a long-lived flow 1 is competing with a short-lived flow 2 which starts and ends at times 1000 and 1250, respectively. We observe (in the zoomed-in bottom plots) how the average value of available rate catches up with the instantaneous reported values as the monitor switches

to the agile EWMA filter, so that the JTP source of flow 1 quickly backs off or increases its rate accordingly. The top plots show the fair convergence of flow rates when flow 2 is present. Note that flow 1 will ramp up close to its original sending rate after flow 2 terminates– the time window that the graph is showing is small and does not show that.

6.4 Statistics Collection

In the previous section, we assumed that the JTP packets carry accurate statistics about the traversed path, which are necessary for monitoring the path. In MANETs gathering accurate network statistics is challenging. The communication is over a shared medium and thus the network conditions are affected by the network density. Moreover the clocks of the hosts are not synchronized and access to NTP servers or GPS is not guaranteed, making even the simple task of measuring network delays challenging. The collaboration of mid-hop nodes is essential in accurate monitoring of the path. Unlike legacy networks, JAVeLEN's architecture allows for easy hop-by-hop packet inspection and modification by any module in the networking stack.

As discussed in Chapter 3, JTP installs within the JAVeLEN MAC the DPS plugin that is responsible for statistics collection and reporting. This plugin implements a mechanism very similar to the Dynamic Packet State mechanism proposed by Ian Stoica $et; al.$ [SZ99] which avoids per-flow state. All the necessary information is included in the packet and the DPS plugin is responsible for modifying the JTP header information at each node. JTP only cares about aggregate path statistics and thus this mechanism scales well with the number of nodes in the path.

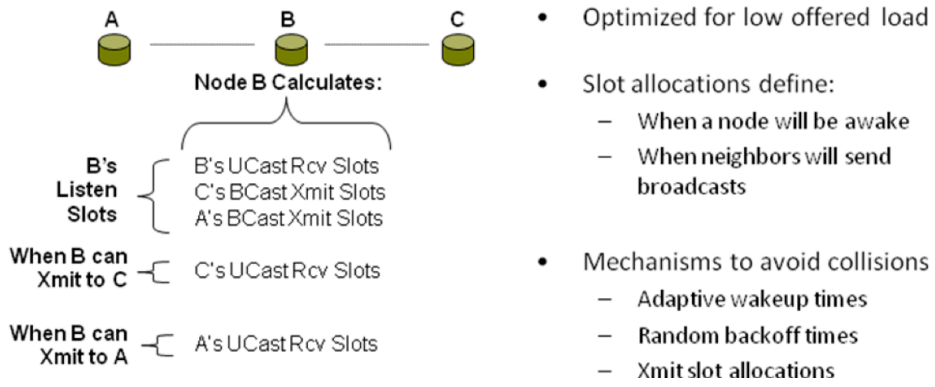
Currently JTP collects the *minimum available rate* and the *one-way delay between the end hosts*. Each of these statistics has its own challenges to monitor and collect. In this section we provide more details about how each of these statistics is computed.

6.4.1 Available Rate Computation

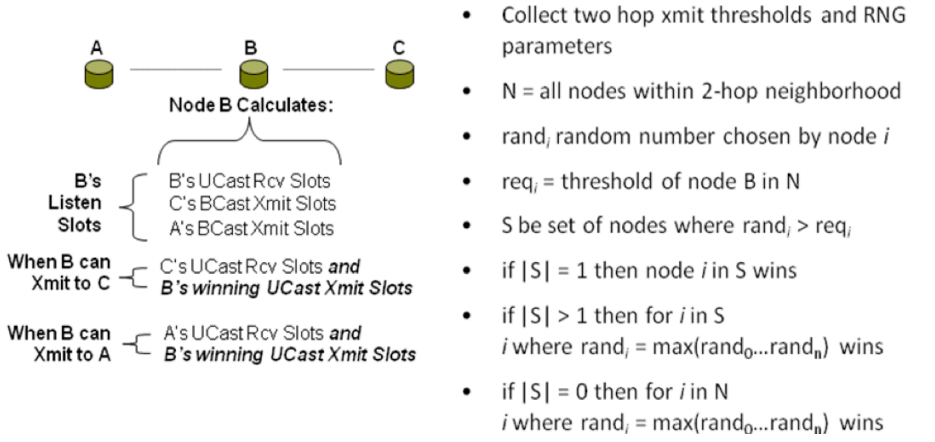
Computing the available rate of a path in a MANET is a very challenging problem due to the shared nature of the communication medium. The number of extra packets that node A can send to node B, depends on how many other nodes are around, and on how busy is A or B sending and receiving other packets. Also if node A increases the rate at which it sends packets to B, this will also increase the number of packets that node B needs to forward further to another neighboring node C.

In the literature there are two main categories of tools that try to measure the available bandwidth in a network: the *probe-based techniques* and the *passive measurement techniques*. Probe-based [SKK03, LPP04] techniques attempt to measure the available bandwidth by periodically sending probe packets in the network. These techniques are not appropriate in a system that strives to save energy since they increase the control packets in the network. Passive measurement techniques attempt the measure the available bandwidth by passively monitoring the network [LHY+06, WFC09] . The reader is encouraged to read this survey [GWMC09] that compares the performance of several of these tools in wireless networks.

JAVeLEN deploys a coordinated slotted MAC [RCP+04], where the nodes periodically turn off their transceiver to conserve energy. For coordination, JAVeLEN uses pseudorandom schedules. The JAVeLEN MAC adapts across the spectrum of offered loads. Under high-offered loads, it performs nearly as well as a dynamic time division multiple access (TDMA) protocol without all the associated overhead that causes problems under mobility. Under low-offered loads, it strikes a performance balance between delay and energy utilization. In low-offered load, the schedule takes into account only the reception slots of each node; if a node is awake for receiving then any node that has a packet for it will transmit. Figure 6.8(a) shows the details of the algorithm that nodes use in low-offered mode. In high-offered load on the other hand, both reception and transmission slots are taken into account; if a node is awake for receiving then only the node that is in transmission mode can send a packet. Figure 6.8(b) shows the details of the algorithm that nodes use in



(a)Calculating wakeup times in low-offered load.



(b)Calculating wakeup times in high-offered load.

Figure 6.8: Algorithms for computing wakeup slots under different loads.

high-offered mode.

JTP computes the available capacity on the path based on the reception availability of each node from the previous hop on the path. In an environment of coordinated schedules, where a node knows the reception status of all neighboring nodes it should be easy to compute the idle slots and convert this into the available capacity of the link. When a node is in low-offered mode it counts how many of its reception slots were idle and reports that as its available reception capacity. Although this capacity is shared amongst all the potential senders, when the load is low this is a good approximation. If there are many

competing senders, the node will eventually transition to the high-offered load mode. In the high-offered load mode a node counts all the slots that are idle not only in reception mode but also when the intended sender for the flow is in transmit mode.

In the current JAVeLEN implementation, the MAC layer over-estimates a node's reception capacity by counting wakeup slots in which hail collisions (losses) happen as idle. Thus, to achieve an ideal collision-free MAC access and assuming back-logged sources, the *effective* maximum available reception capacity of a node is given by the maximum available reception capacity divided by the number of nodes in the collision (two-hop neighborhood) domain of the source–for example, in a linear topology of five nodes, say A-B-C-D-E, a C-D transmission is only successful if all other four nodes, A, B, D and E, do not transmit. Thus, the reception capacity of D is effectively its maximum capacity $MaxCap$ divided by the size of its two hop neighborhood (N) which in this case it is 4. This effective maximum available capacity determines the feasible range of available rate [δ, $MaxCap$]–below this value of δ, sources should back off.

6.4.2 One-Way Delay Computation

The one-way delay is computed using relative timestamps. Relative timestamps is a popular method for measuring time in MANETs and DTNs. This is because in these environments keeping time synchronization between nodes is hard, since communication between nodes is not guaranteed, GPS is not always available and there is no back-channel to a central server that could help in time synchronization. Relative timestamps work as follows:

- When a packet is created it is stamped with the current value of the host's clock.
- Before the packet goes out over the radio, the timestamp is compared against the current time to compute how much time has elapsed from the time the packet was created till now, and the absolute difference replaces the timestamp.
- When a packet is received at the next hop, the time the packet spent in flight is added to the current value of the timestamp, and this aggregate value is subtracted from the

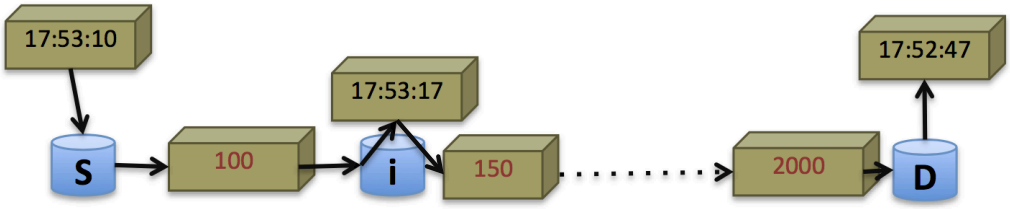


Figure 6.9: The value of the timestamp field as the packet moves through the network.

current value of the clock. The result of this is the clock value of the current host at the time that the packet was generated. Although the clocks between hosts are not synchronized the propagation delay is usually well known, and thus the time a packet is in flight can be easily computed by the MAC.

- At the final destination the timestamp is converted to an absolute time upon reception, just like in any other receiving nodes. When the packet arrives at JTP this absolute time is used to compute the time it has passed from when the packet was created until it was received by the JTP module of the final destination.

Figure 6.9 shows an illustration of the above method. When the packet is moving between nodes, it carries the relative time that has passed since the packet was created. When the packet is processed within a node it carries the absolute value of its creation time according to the local clock. When a packet reaches its destination, the host can compute how much time has elapsed since the packet was generated at the source. Let's consider the packet in the figure as it traverses the first hop. At creation time the packet is timestamped with the current value of the clock, i.e. $17:53:10$. Before the packet is transmitted, the sender subtracts from the current clock value the original timestamp resulting in a relative timestamp of $100msec$. When the packet is received at the second hop – node i – the relative timestamp is increased by the time the packet was in flight and the aggregate value is subtracted from the current value of the clock to compute the value of the local clock, when the packet was created. In this example the clock at node i was $17:53:17$ when the

packet was created at node s. This process is repeated at each hop and thus the destination can compute the time of packet creation based on its local clock. Note that the creation time for the same packet differs between nodes due to clock skew. For example for the destination the packet was created at time $17:52:47$ while for the source the packet was created at time $17:53:10$.

6.5 Conclusion

In this chapter, the flow control mechanisms of JTP were presented. JTP is a rate-based protocol that uses a PI^2/MD controller in order to set the transfer rate based on the available rate measurements. JTP's rate controller combined with a flip-flop filtering mechanism allows JTP to quickly adapt to significant network changes without reacting to transient variations. For the purpose of this thesis JTP has been implemented within the JAVeLEN system and thus gets the available rate computation from the MAC. However, if JTP was deployed on top of a different system that did not provide such information, JTP can employ one of the existing tools available for wireless networks to estimate the available bandwidth.

In this chapter we also presented how caching and in-network retransmissions can inadvertently affect the fair allocation of bandwidth between flows. JTP employs a back-off mechanism at the sender to ensure that flows share the resources evenly.

At last, we presented the variable rate feedback mechanism. This mechanism of JTP draws inspiration from the delayed ACK mechanism of TCP, but further enhances it since the rate of feedback packets in JTP is not instrumental in setting the transmission rate. JTP strives to minimize the control traffic while still being responsive to network changes and while ensuring that the application meets its requirements.

Chapter 7

Implementation and Experimental Results

In order to validate JTP premises, and demonstrate the feasibility of the design we implemented JTP and tested it within the JAVeLEN system. JTP is implemented using an Operating System Abstraction Layer (OSAL), developed by BBN which allows for the same code to be used over multiple platforms. In this chapter, we present the implementation details and discuss in detail the experiments we ran both in simulation and in real deployments.

Implementing and testing JTP within the JAVeLEN system allows us to validate our analytical results. As we are going to see later in this Chapter, the results from both simulations and real deployments validate our analysis. JTP outperforms other state-of-the-art approaches, and is able to satisfy a range of application requirements while conserving energy.

The implementation of JTP happened side by side with the design, ensuring that the architecture is feasible and realizable on an embedded system, like the JAVeLEN radio. Coding all of JTP mechanisms early in the design process, guaranteed that any implementation hurdles are accounted for. A good example is the fact that in most embedded real-time operating systems, floating-point arithmetic is not supported. The JAVeLEN MAC being a slotted protocol, needs to run on a real-time OS and thus all operations of iJTP– see Section 3.2.2 for more details– need to be kept simple and any more complicated operation

need to be implemented with lookup tables.

Another challenge that we faced while implementing iJTP was that all operations that happen within the context of a slot need to be tightly time-bounded or else there is the danger of blowing a slot. A slot is blown when the time for processing and transmitting data is longer than the slot duration and thus the slot does not finish on time. This really constrains the amount and type of operations that can run within the iJTP context. Moreover at the design phase it was important to ensure that all iJTP algorithms have strict running time bounds and are preferably constant-time algorithms.

In summary, the implementation of JTP played an important role in its architecture and final design. It helped keep our ideas grounded, and was always the feasibility compass that ensured that JTP mechanisms can be implemented on a real system. Moreover using BBN's OSAL, we were able to use the same code in simulation and in the actual system enabling us to validate our ideas and analytical results and showcase JTP premises as well as demonstrate how JTP outperforms other approaches in energy constrained mobile environments.

7.1 JAVeLEN Implementation Environment

Developing, evaluating and testing tactical communication systems is very challenging, due not only to limited availability of hardware and wireless testbeds but also due to costs of manpower for running experiments. Although simulations cannot replace real-world experiments, their use is vital in designing and analyzing the network behavior under various circumstances. Usually the software written for simulation testing is separate from the code running on the target platform and the two code bases become divergent over time and it becomes hard to keep bug fixes and protocol updates consistent between them. Over time the actual protocol behavior modeled and simulated is different from the protocol behavior in the field. In order to address this complex problem BBN has developed a software framework called Portable Link Framework (PLF).

PLF is an API that enables implementations that can be easily ported between different systems. PLF may be thought of as an Operating System Abstraction Layer (OSAL). PLF provides inter-thread and inter-process communication primitives, as well as configuration and data collection primitives, memory management, string manipulation, random number generation, and other critical operating system functions which cannot be guaranteed to be the same across various real-time, non real-time and simulation platforms. PLF has been ported to multiple systems already (Linux [lin], RTLinux [Yod99], Integrity [int], OPNET [opn], etc.)

All JAVeLEN protocols, including JTP, have been implemented on top of PLF ensuring that the same code is used in simulations and in the various target platforms that have been used during the deployment of the system. This shared code model lets us test algorithms before the target platform is available and explore the scalability aspects of the system beyond what the target platform can support. Porting to a new platform becomes an easier task as all that needs to be written is an OSAL for that platform. Moreover using the same code over different platforms makes it easier to compare simulation results with real implementations since most of the code used is the same and thus removes any artifacts in performance due to specific implementation of the used algorithms.

A valid concern is that the performance of the system using an OSAL implementation will not be as efficient as a native implementation might be. Although an implementation over an OSAL, will probably always contain some inefficiencies compared to a native implementation, PLF's implementation is designed carefully to be as efficient as possible, while maintaining a useful abstraction. To this end the PLF environment provides:

- support for multi-threaded environments with preemption and no thread reentrancy. This allows the application designers to break down the application in multiple threads and assign different priorities to them based on how critical are the tasks that are performed by each thread,

- zero memory copies at the framework level, which leads to very efficient memory

management at least within the framework, and

- fast context switch, and event handling guarantees, which allows applications to run in real-time OS were hard-time guarantees are needed,

Although a complete native implementation in each one of the targeted platforms can be at least as efficient as the PLF implementation of the same application, the benefits of the sharability outweighs any benefits from a native implementation. Allowing the programmer to use the same code over all platforms, enables more robust and well designed implementations that in practice might be more efficient than multiple native implementations on all the different supported platforms.

7.1.1 JTP Implementation

The JTP implementation follows the "shared-cod" model of the JAVeLEN system. The core of JTP mechanisms is coded on top of the PLF API enabling the same code to be debugged and run in all environments (OPNET, RTLinux, Integrity). As described in Chapter 3, JTP has two main components; eJTP and iJTP.

iJTP is part of the JAVeLEN MAC and runs as a plugin. iJTP only interacts with the Link Characterization module of the MAC and all its code is shared.

eJTP on the other hand, needs to interact with the applications as well as with the underlying infrastructure. PLF was designed to support the implementation of different Link Layer protocols and thus was missing abstractions that would help in the implementation of a transport protocol. We made a JTP specific extension of the OSAL to incorporate system calls specific to JTP. Most of the functionality that needed to be abstracted was socket related and was necessary for the Linux kernel implementation of eJTP.

eJTP's functionality encompasses the interaction with applications and with the lower layers of the protocol stack where other JAVeLEN protocols reside, including iJTP. This is the OS-specific implementation of eJTP. If eJTP is implemented as a Linux Kernel Module for example, this interface is mainly provided by the socket infrastructure. The different

eJTP modules depicted in Figure 3.2 might be fully or partially implemented in shared code. Figure 7.1 depicts the implementation of the various modules.

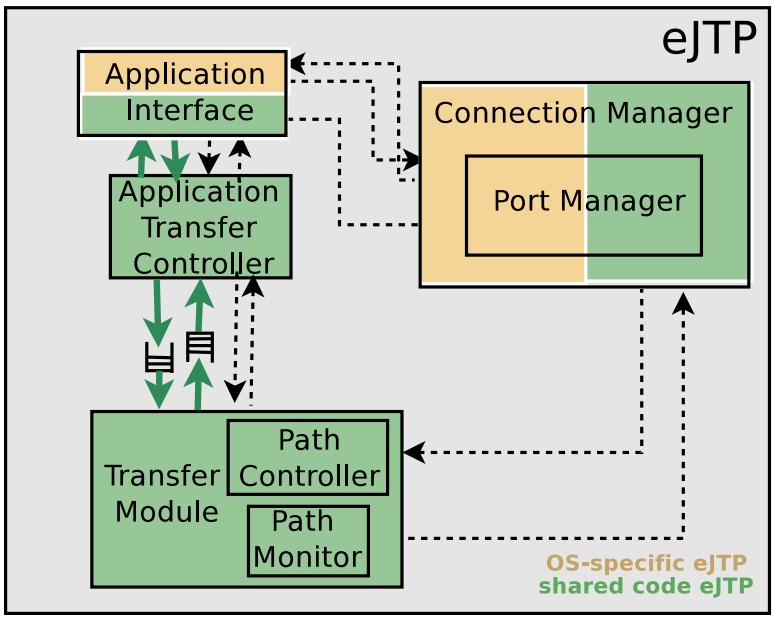


Figure 7.1: Schematic view of eJTP's implementation.

Figure Figure 7.2 depicts a schematic view of the OSAL for eJTP. This OSAL involves the following parts:

1. *Memory management*: wrapper functions to native memory management routines.

2. *Packet management*: routines to handle the notion of a packet as it is defined in the underlying platform being used, converting it to the notion of a packet as seen by JTP's shared code.

3. *Connection management:* routines to provide uniform access to the platform-specific connection information.

4. *Timer management:* Wrapper functions to enable eJTP to use the infrastructure of timers provided by the underlying platform. These wrapper functions can be modified to implement a shared eJTP implementation of timers if the underlying platform does not provide one.

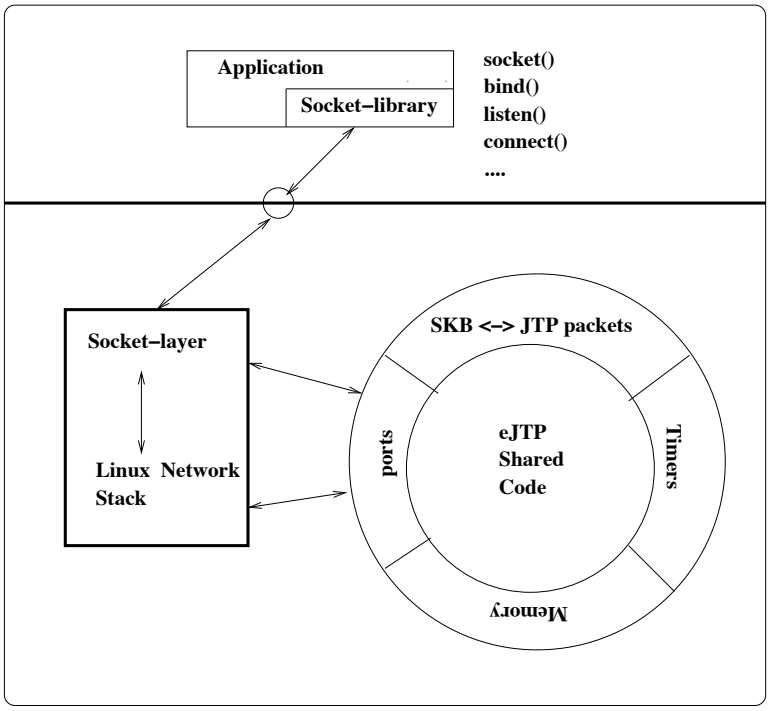


Figure 7.2: OSAL components in eJTP implementation. Linux is shown as the underlying platform.

5. *Port management:* Wrapper functions to enable eJTP to use the infrastructure of port management provided by the underlying platform. As for timers, these wrapper functions can be internally implemented so that eJTP provides its own port management.

There are multiple OSALs for the JAVeLEN system and by extension for JTP. There are JTP implementations for OPNET, user level Linux, Linux Kernel Module, RTLinux and Integrity. For simulations we used the OPNET simulator, while the final product was implemented in an embedded device running Linux and Integrity operating systems.

7.2 Performance Evaluation

Earlier in the thesis, we have shown simulation results that demonstrate the operation and performance of the various mechanisms in JTP. In this section, we provide all the simulation parameters that we used for the experiments, we describe in detail the algorithms we compare JTP against and we present all the experimental results both from simulation and from real deployments.

7.2.1 JTP and other Protocols

In order to provide a comprehensive and fair comparison against existing transport approaches for multi-hop wireless networks [LS01, SAHS03, CNV04, ZCF05], JTP is compared against a representative set of protocols. JTP is implemented within the JAVeLEN system and thus has all the benefits of being fully integrated and has access to statistics provided by the MAC. To ensure that we have a fair comparison with other protocols, we modified our code so as to implement the protocols we are comparing against. In this way the protocols are tightly integrated with the rest of the system and we used the same kind of statistics that JTP has access to when needed. Moreover with this approach we ensure there are no artifacts in the results due to separate implementation, since most of the code is shared between protocols.

JTP is compared against:

TCP-SACK

TCP-SACK [MMFR96] is chosen as a representative for all window-based approaches and as the most commonly implemented protocol in working systems. In order to have a more competitive performance, we use a rate-based flavor of TCP-SACK, whereby the rate of each flow is set by the well-known throughput equation of TCP [PFTK98]. Specifically the rate is set as:

$$\frac{1}{RTT \times \sqrt{\frac{2bp}{3}} + RTO(3\sqrt{\frac{3bp}{8}})p(1+32p^2)} \tag{7.1}$$

where:

- RTT is the round-trip in seconds, which we estimate the same way we estimate the one-watt delay for JTP,
- RTO is the retransmission timeout; for simplicity we set this as $4xRTT$,
- b is the number of packets acknowledged by a single TCP ACK – we use delayed ACKs and thus $b = 2$ in our simulation,
- p is the packet loss rate computed at the sender of the connection.

Using a rate-based equivalent TCP, we remove artifacts from the window-induced burstiness of data and ACK streams, similar to what TCP pacing [ASA00] does. We use Selective ACKnowledgments so that only lost packets are retransmitted.

ATP-like

Except for window-based approaches, there is a whole suite of explicit rate-based transport protocols. In order to compare against them, we implemented a representative protocol based on the ATP [SAHS03] approach. The implemented protocol:

- adjusts the sending rate based on explicit feedback collected by intermediate nodes,

- supports only end-to-end recovery,
- supports only full reliability, and
- has constant-rate feedback from the receiver – as suggested in the paper the feedback period is set to be larger than RTT.

ATP adjusts its sending rate based on explicit rate feedback from the path and thus takes into account real congestion, capturing the behavior of TCP CLAMP [AHM03].

UDP

Since both TCP-SACK and ATP both support only reliable transfers, we also use UDP as a base line when running semi-reliable experiments. UDP has no reliability mechanism and thus it just reflects the base reliability of the path. UDP is easily implemented as JTP with 100% loss tolerance.

JNP: JTP No Caching

In order to evaluate the benefits of JTP's caching mechanism, we compare JTP against a JTP variant, which basically includes all JTP mechanisms except the caching one, i.e. the retransmissions originate from the source. This comparison enables us to isolate the benefits of caching and study better its behavior. Performance results in Chapter 6 used JNC for comparison.

7.2.2 JTP Parameters

In this section we provide a detailed list of JTP parameters that were used in the experiments. We have tested JTP using the OPNET simulator (Section 7.2.4), within an early JAVeLEN prototype system (Section 7.2.5) and in the final JAVeLEN radios (Section 7.2.6). Although specific details for each of these tests are presented in the related section, Table 7.1 presents the default parameter values for JTP, so unless otherwise stated, these default values are used throughout the experiments. In this prototype implementation the

JTP header is 28 bytes and the JTP ACK header is 200 bytes. The JTP headers, and especially the ACK headers are not optimized in this prototype implementation.

7.2.3 Performance Metrics

In this section we discuss the performance metrics that are used for comparing JTP with other protocols. Given that TCP-SACK and ATP only support 100%-reliability transfers, we will consider only bulk transfers with 0% loss tolerance.

The metrics used to compare the efficiency of each protocol are:

- *Energy per delivered bit*: This measure captures the system-wide energy consumed to deliver each data bit to applications. Since our goal is to evaluate the energy consumption of transport protocols, we are only concerned with the energy actually spent to transfer packets of the transport layer, and thus we will not consider the energy consumed for network maintenance by the lower layers. In the JAVeLEN system, the Link Layer is responsible for specifying the transmission power, and the datarate of the radio on a per packet basis. Based on the link layer settings and the length of the packet we are able to get a very good estimate of the energy spent for a specific packet. We place a monitor at the link layer that computes the energy spent for the transmission of each transport-layer packet and get aggregate statistics about the total energy spent. Moreover in real deployments we install a probe in some nodes that will measure the actually energy spent during a run.

- *Goodput*: This measure captures the total rate at which the network delivers new data to the applications, and thus represents how efficient the network resources are utilized.

7.2.4 Simulations in OPNET

We have used the OPNET [opn] simulator in order to test the performance of JTP in different topologies and scenarios. Using an OPNET PLF OSAL implementation we run

Table 7.1: Default parameter values

Parameter	**Value**	**Description**
General Parameters		
MAX_ATTEMPTS	5	Maximum number of link-layer retransmissions
MAX_PROBES	5	Maximum number of probes before a connection is declared down
JTP Pkt Size	800 bytes	Maximum JTP packet size
Cache Size	1000 pkts	Cache size of mid-path nodes
T_{Lower_bound}	10s	Minimum duration between two consecutive feedback pkts
Rate Controller		
MIN_RATE	6,000bps	Minimum rate of a JTP connection
MAX_RATE	500,000bps	Maximum rate of a JTP connection
Decrease Constant	0.7	Constant value for the multiplicative decrease
Decrease Mode Threshold	5%	Percentage of the maximum rate, that when the available rate falls under, the controller goes to decrease mode
Path Monitor		
Stable α	0.09 (0.15)	α parameter for the EWMA of a metric while the controller is in stable state, all controllers use the first value except from the OneWayDelay controller that uses the value in parenthesis
Stable β	0.01	β parameter for the EWMA of a metric while the controller is in stable state
Stable δ	1.28	Weight for upper/lower limits for a metric
Stable MAX_OUTLIERS	3	Number of consecutive outlier reports before switching to the agile controller
Agile α	0.6	α parameter for the EWMA of a metric while the controller is in agile state
Agile β	0.6	β parameter for the EWMA of a metric while the controller is in agile state
Agile δ	1.28	Weight for upper/lower limits for a metric
Agile MAX_INLIERS	3	Number of consecutive inlier reports before switching to the stable controller

the whole JAVeLEN stack in the simulator and we use a traffic generator to produce traffic. In order to determine the quality of the link between nodes we use pathloss files that define the pathloss value for any link in the network over time. In OPNET we used three different types of topology:

- **Static Linear Topologies.** We use static linear topologies to study the effect of path length between sender and receiver on performance. The sender is placed at the first node in the line while the receiver is placed at the last node. We use a fixed drop-off communication model where if the distance is greater than a certain value then two nodes cannot talk to each other. In these topologies each node can talk to its adjacent nodes in the line but cannot talk to the rest of the nodes. We used these controlled topologies to study different aspects of JTP flows in a repeatable controllable environment.

- **Static Random Topologies.** We use these topologies to study the effect of node density, cross-flow interactions and route symmetry. To generate these pathloss files we first randomly place the specified number of nodes in a pre-defined bounding box, and then use the distance between nodes to compute the quality of the links. We use a pathloss model where pathloss is proportional to the 4^{th} power of distance [Gol05], more specifically to compute the average pathloss between two nodes we use the equation:

$$P(x, y) = 16\pi^2 d(x, y)^4 \tag{7.2}$$

  where $P(x, y)$ is the pathloss between nodes x and y and $d(x, y)$ is the Euclidean distance between the nodes.

- **Mobile Random Topologies.** We use these topologies to study the effect of mobility. To generate the pathloss files for these topologies, we first create a random static topology and then use the random waypoint model [JM96] to compute the new

locations of the nodes at each time step. After the locations have been determined, Equation 7.2 is used to compute the pathloss values between nodes. In the random waypoint model, each node can either be on the move, or be static. If a node is static then the duration of the pause is determined at random based on the mean pause time, while if a node is moving the length of the path is determined by the average path-length defined. The speed of the node also follows a Normal distribution for given mean and variance values.

Results in Static Linear Topologies

In these experiments the source and the destination of two competing flows are placed at the two ends of the network. To capture the varying quality of wireless links, the value of the average *pathloss* of each link alternates between a good state (low loss) and a bad state (high loss). Each link is in bad state approximately 10% of the time. The average duration of the bad period is 3 seconds. The results shown are the average of twenty (independent) runs along with 95% confidence intervals. Each simulation run lasted for 2500 seconds, and flows were started randomly after a warm-up period of 900 seconds.

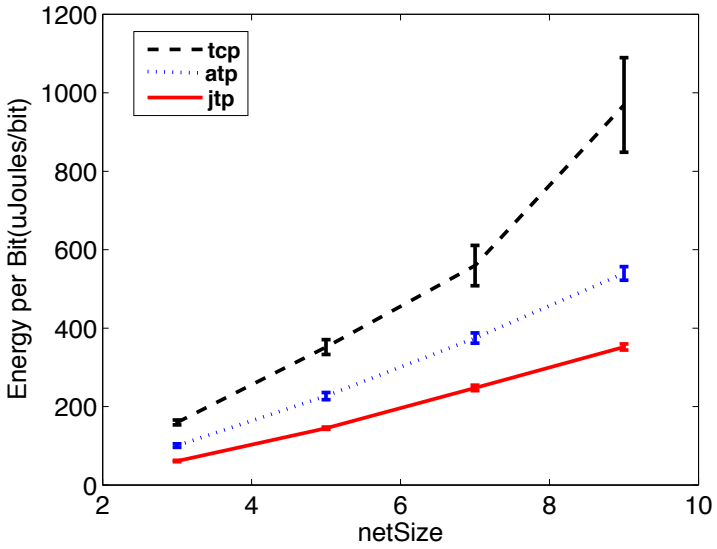


Figure 7.3: Linear: Total energy expended per application data bit delivered.

Figure 7.3 shows the energy per delivered bit for each protocol for varying network sizes. JTP significantly outperforms all other protocols. As the path length increases, ATP ends up expending twice as much energy as JTP to deliver one bit, while TCP-SACK expends almost five times more energy for the delivery of each bit.

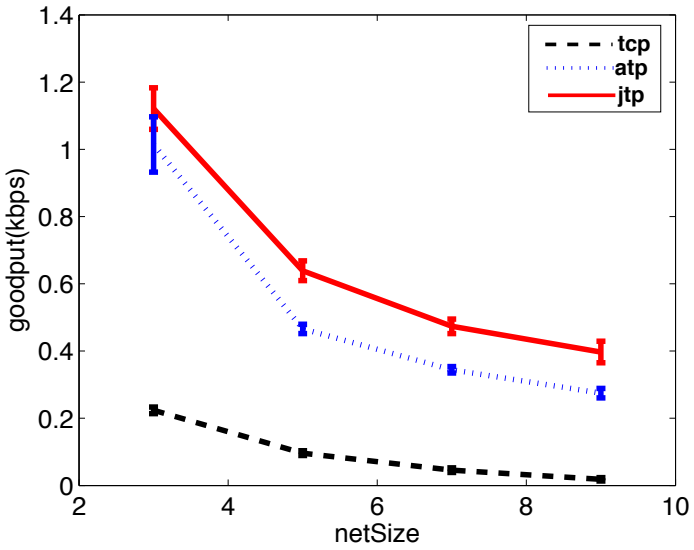


Figure 7.4: Linear: Average goodput experienced by flows in the network.

Figure 7.4 demonstrates that JTP not only provides great energy savings, but also achieves higher goodput. Without sacrificing system's performance, JTP minimizes the amount of feedback control messages, which in a wireless network environment, effectively "steal" bandwidth from users' data.

Results in Static Random Topologies

We also tested JTP over static random topologies. Nodes are randomly distributed in a two-dimensional field. In order to avoid getting disconnected topologies, the field size is set to ensure that the network is connected with high probability. The source and destination nodes of five simultaneous flows are chosen randomly. The presented results are the average of 10 independent runs, of 4000 seconds each, both for the static and the mobile scenarios. Given that the placement of the nodes and flows are chosen at random, the system-wide

performance might vary significantly. In order to meaningfully compare across different protocols, we ensured that all the protocols run under the same conditions in the same run.

Figures 7.5 and 7.6 show the energy per delivered bit and the goodput achieved by various protocols for varying network size. JTP outperforms both ATP and TCP in both metrics.

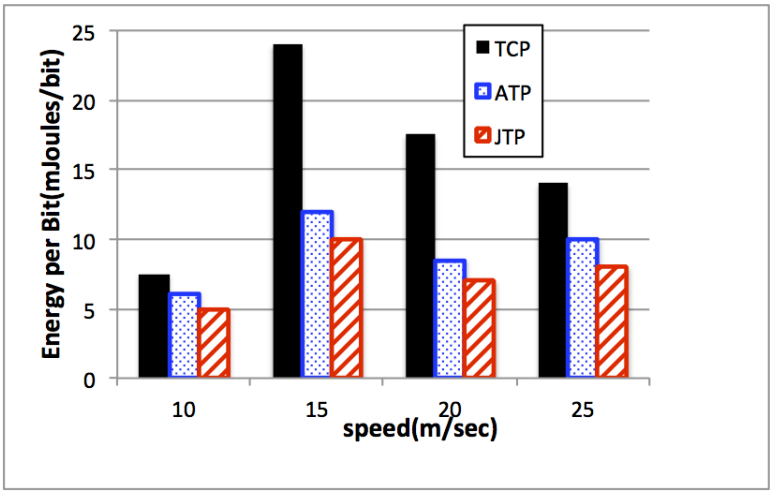


Figure 7.5: Random: Total energy expended per application data bit delivered.

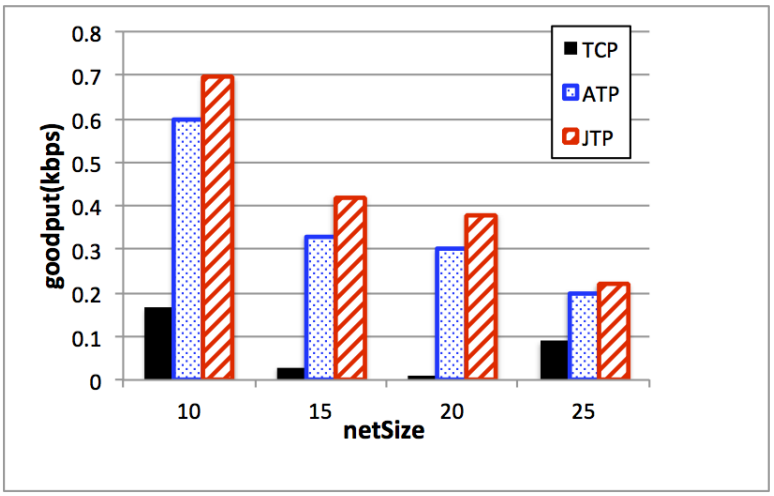


Figure 7.6: Random: Average goodput experienced by flows in the network.

Results in Mobile Random Topologies

In the next set of experiments, we tested JTP's performance in a mobile setting for a 15-node network. Each node moves within the field at various speeds (low: $0.1 meter/second$, moderate: $1 meter/second$, fast: $5 meter/second$). We used the random way point mobility model in which each node chooses a random direction and moves in that direction for an average distance of $47 meters$. There is an average pause of 100 seconds between movements for each node.

Figures 7.7 and 7.8 show the energy per delivered bit and the goodput achieved by the three protocols. JTP outperforms both ATP and TCP in both metrics. Figure 7.9 presents the relation between end-to-end and locally recovered packets—the values presented are normalized by the total data delivered to the applications. This graph shows that even in mobile environments, where the path between two nodes is constantly changing, deploying local caches is beneficial—we observe in-network retransmissions which result in energy gains and better distribution of retransmission effort across nodes.

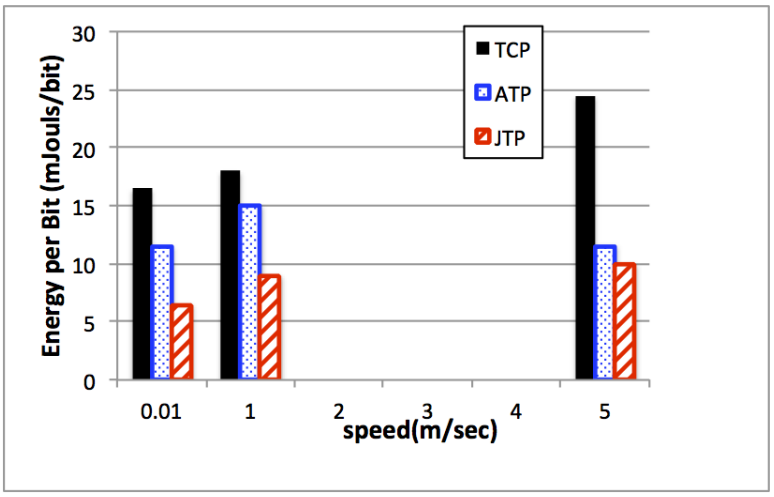


Figure 7.7: Mobile: Total energy expended per application data bit delivered.

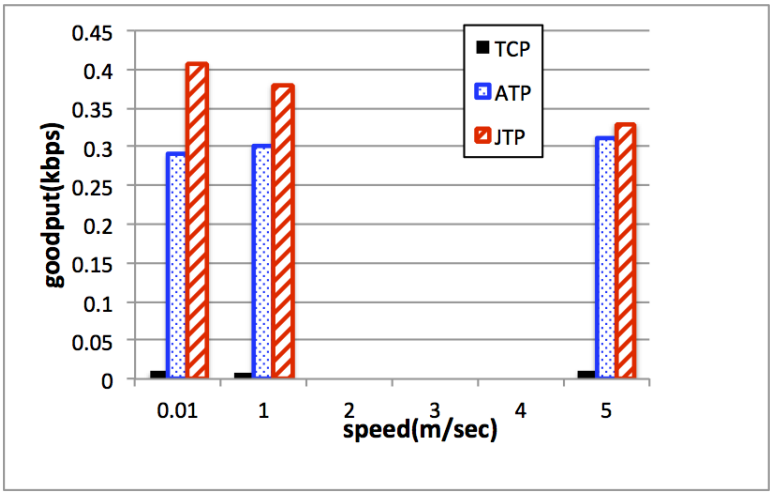


Figure 7.8: Mobile: Goodput delivered to applications.

7.2.5 Results in Prototype JAVeLEN Radios

The first prototype JAVeLEN system was built within custom boxes which were handmade at BBN. Figure 7.10 shows a picture of such a node. In our setup there are thirteen nodes spread within offices of different floors of a BBN building. Figure 7.11 shows the discovered topology of the nodes.

The prototype nodes have two processors, one with RTLinux where the JAVeLEN MAC runs, and the other with Linux where JavRoute and JTP run. The nodes are also equipped with two antennas, one for the Hail and one for the Data radio.

In order to verify our simulation results, we implemented and tested JTP in a real JAVeLEN system. In this system the non real-time part of the system, like the applications, are running on top of Linux. In these experiments we ran JTP, TCP-SACK and ATP. Each experiment lasted for 30 minutes. During these 30 minutes, flows were generated in each node with an average inter-arrival time of 400and average transfer size of 100KB. A summary of the results is shown in Table 7.2. Given that in the real system the pathloss of the links is not controlled but only determined by the in-door multipath fading, the links are more stable and their quality is much better, which results in lower energy consumption for all protocols. Nevertheless, JTP still outperforms both ATP and TCP in both metrics. With

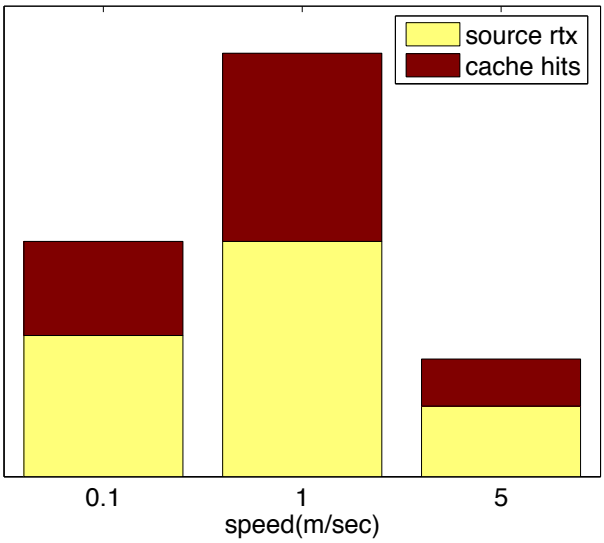


Figure 7.9: Mobile: Relation between end-to-end and locally recovered packets.

Table 7.2: JAVeLEN system results

	Energy per delivered bit (mJ/bit)	Average goodput (kbps)
JTP	0.0054	0.63
ATP	0.0068	0.44
TCP	0.0105	0.17

respect to energy JTP provide 20% improvement as compared to ATP, while it consumes half the energy consumed when using TCP. JTP also provide a 50% improvement in goodput as compared to ATP while it triples the goodput experienced by the applications when TCP is used. Notice that the goodput achieved by TCP is higher than that achieved in simulation due to the low packet loss rate.

7.2.6 Results in Final JAVeLEN Radios

The final JAVeLEN radio is based on ARLs 2nd Generation Blue Radio platform. The platform shown in Figure 7.12 has dimensions of 3 inches by 3 inches by 1.4 inches, accepts 6-17 volts input power and contains three boards:

1. The RF transceiver board which can support fast T/R (transmit/receive) switching

Figure 7.10: JAVeLEN prototype node.

times and it can wake up from a low power (i.e. idle) state quickly.

2. A custom modem board that contains a low power Analog Devices Blackfin high performance Digital Signal Processor (DSP) to run the software modem code.

3. The networking and external interface board that contains a low power Freescale iMX.31 to run the MAC, Routing and Transport protocols.

Although in the final demonstration it was hard to isolate and test the performance of just JTP, the results presented in this section validate JTP's premises in real-life scenarios, and demonstrate its usability in actual systems. JTP was not only able to enhance the energy savings achieved by the rest of the JAVeLEN system, but also allowed the nodes to quickly adapt to changing network conditions and enabled efficient high-bandwidth transfers.

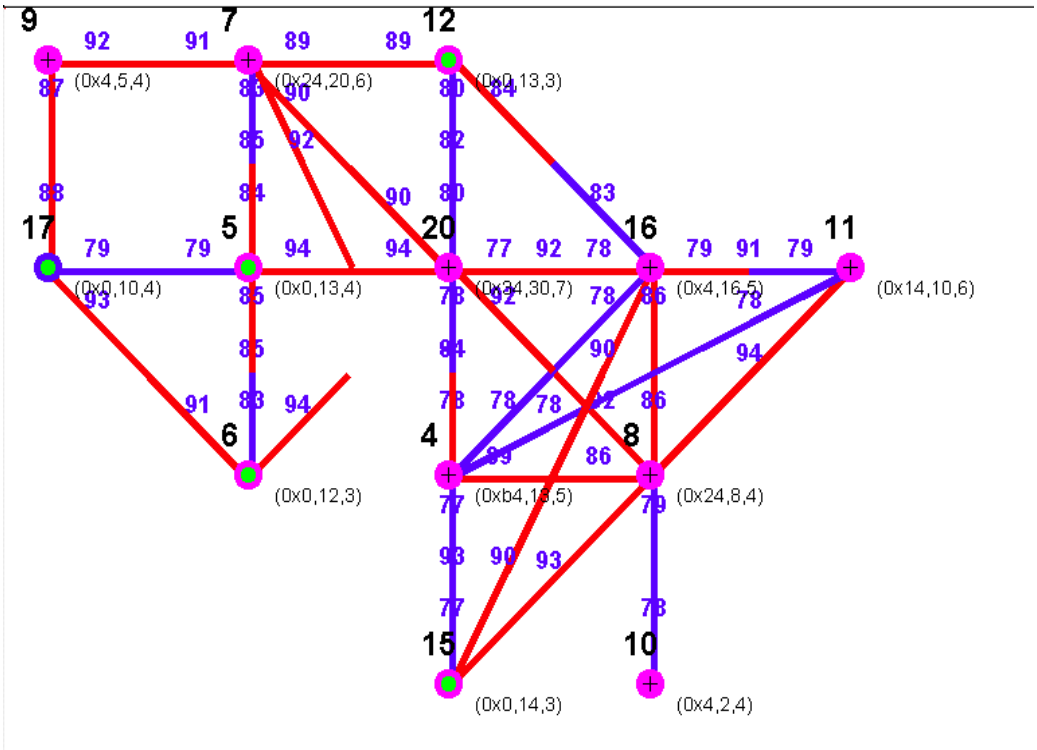


Figure 7.11: Topology of BBN original testbed.

In the final demonstration, the whole JAVeLEN system was compared against a baseline system of 1Mbps 802.11 with OLSR and a one watt power amplifier. The two systems were compared in two different scenarios:

1. **Roadside Monitoring Scenario.** In this scenario passive infrared (PIR) sensors with imaging capability are placed by the side of a road to monitor activity. When movement is detected the cameras of the sensors are turned on and start transmitting video back to the Tactical Operations Center (TOC). The number of frames and the resolution of the clips were variable - sensors were configured to send between 3 and 10 frames per clip with up to 320x240 resolution.

2. **Remote Building Surveillance Scenario.** In this scenario a building is monitored by PIR sensors. When movement is detected the triggered sensors start capturing video and send aggregate statistics back to TOC over a satellite link. A vehicle is sent

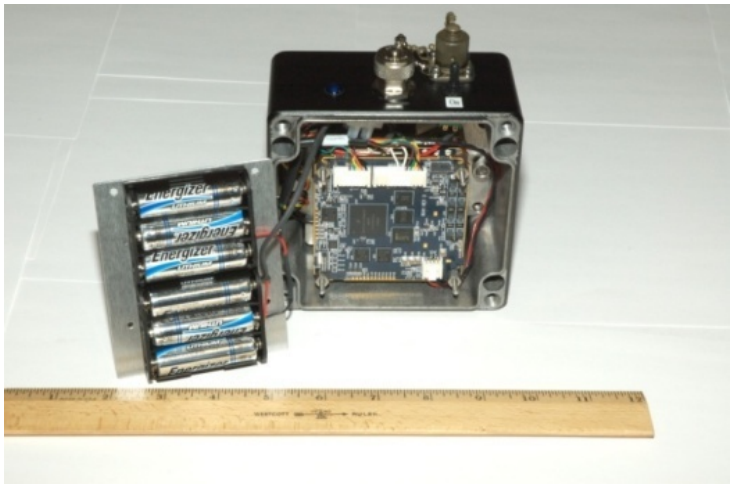


Figure 7.12: Final JAVeLEN radio.

to collect the image data and then transfer them back to TOC over the JAVeLEN network. In the scenario traditional store-and-forward techniques were used to transfer the data over the disconnected network and JTP was used as the underlying transport protocol for the communication.

Figure 7.13 shows a real-time view of the energy spent on a node, and we can see that the JAVeLEN system achieves energy improvements of up to 30X over the baseline. The JAVeLEN system not only achieves low energy consumption but is also able to transfer large pieces of data efficiently using JTP. Figure 7.14 shows the expected lifetime of a JAVeLEN node based on the offered load for different battery technologies.

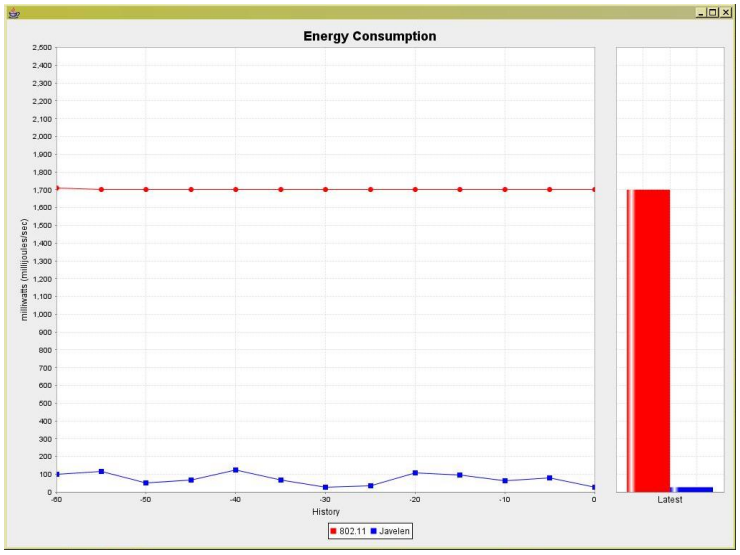


Figure 7.13: Real-time energy display during final demo.

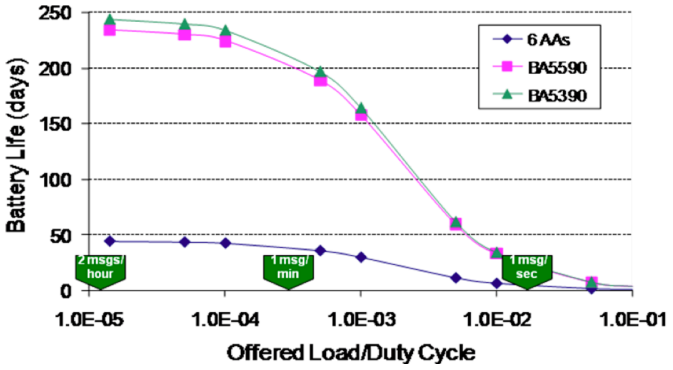


Figure 7.14: JAVeLEN node lifetime comparison.

Chapter 8

Conclusion

In this thesis we presented JTP, an energy-conscious transport protocol for Mobile Ad Hoc Networks. JTP has been implemented as part of JAVeLEN, which is an ultra-low, energy-conserving MANET system. Given an extremely energy efficient media-access layer, JTP has shown, both in simulation and over an operating JAVeLEN network, that it preserves the energy savings of JAVeLEN while maintaining the necessary application performance. This is a substantial accomplishment and confirms that the features we chose are sufficient to achieve energy efficiency.

Chapter 1 of this thesis provides the motivation for this work as well as a summary of the results and contributions. Nodes in a mobile ad hoc network are usually battery operated and thus it is important to conserve energy in order to maximize their lifetime. There has been a lot of research in how to reduce the energy consumption of protocols for various layers of the networking stack in isolation. In JTP we designed a transport protocol within an already energy-conscious architecture, JAVeLEN, striving to increase the energy efficiency of the whole system.

Chapter 2 introduces the necessary background information for the reader to understand and evaluate the rest of the thesis. We provide an overview of the JAVeLEN system and present the key operations that are critical for JTP. The plugin mechanism of JAVeLEN at the link layer gives access to JTP for inspecting and modifying in-flight packets, while

maintaining the efficiency of fast-path forwarding. Moreover the JAVeLEN MAC publishes statistics about the network (packet loss, available bandwidth, etc) to any module in the system, enabling JTP to make decisions based on accurate and timely network information.

In Chapter 2, we review the literature and discuss related research. The role of the transport protocol is instrumental in computer communication and has thus been studied since the beginning of computer networks. Although the challenge of energy-efficient wireless networks is a problem of the last decade,the design of JTP is inspired and founded on a lot of great earlier work.

The detailed architecture of JTP is presented in Chapter 3, where we explain and found all the design decisions. JTP is rate-based and receiver-oriented with a variable feedback mechanism. JTP performs coordinated in-network error recovery, and application-aware per packet handling. JTP is split in two basic modules:

- **eJTP:** that only runs on the end host and is responsible for the establishment and integrity of the data transfer
- **iJTP:** that runs on all hosts in the path and is responsible for inspecting and modifying in-flight packets, as well as for the in-network caching. iJTP is in the fast-path and its functionality has to be kept simple especially in a slotted system like JAVeLEN, where any fast-path operation has to be executed within the duration of a slot.

JTP is designed to support a wide variety of applications by distinguishing the functionalities that are common in any data transfer from those that are application-specific. JTP has a module (ATC) that is placed between the applications and the core transport functionalities and is responsible of fine-tuning JTP based on the application requirements. In Chapter 4 we provide a detailed design of ATC and present results from two different ATC modules: a file-transfer and a VoIP module. Each module is based on the specific application requirements.

In Chapter 5 we present the error control mechanisms of JTP. JTP deploys soft-state packet caches along the path of a connection. These caches enable JTP to recover from lost

packets as close to the destination as possible and thus to conserve the energy of unnecessary transmissions. In order to avoid unfair rate allocation between flows of different reliabilities, JTP deploys mechanisms at the sender to ensure that each flow has the appropriate number of packets in flight.

Chapter 6 describes how JTP adjusts the transfer rate for each flow. The control for the transfer rate lays at the receiver, which monitors the condition of the path and gathers the information that is encoded in the data packets. Based on the collected information it adjusts the rate of the transfer and communicates the new value to the sender. JTP also employs variable-rate feedback to decrease the control traffic. The receiver sends a feedback packet when the path conditions varies significantly or in order to request packets for retransmissions. The results show that JTP reacts faster and sends less packets than periodic feedback mechanisms.

Chapter 7 presents the experimental results. Our results show that JTP outperforms traditional transport protocols (TCP and UDP) and a representative multi-hop wireless transport protocol (ATP) on every aspect. For all network sizes and mobility settings evaluated JTP consistently provides higher goodput and consumes less energy per node and over the entire network.

Our research also contributes in the form of lessons learned. For example, we have demonstrated the need to have multi-level error recovery, via the MAC and caching, that is explicitly controlled by the ends of a connection. This shows that problems in energy-awareness can yield non-intuitive solutions.

JTP provides a fresh perspective that offers many opportunities for future work. We are currently investigating energy-awareness in cache/memory management, and the support of other applications such as data collection and image transfers.

Appendix A

JTP interfaces

A.1 JTP Interface to the Applications

JTP provides an API to the applications, see Chapter 4. The functions described here are part of the shared code of eJTP. Within the JAVeLEN architecture JTP runs as a separate process or thread and thus will only provide non blocking calls as interface to the application. Depending on the platform where JTP is to be deployed a platform specific part should be implemented that will wrap these functions based on what the application is expecting. We tried to follow the socket API as it is implemented in Linux, in order to make it easy to port JTP as a kernel module.

A.1.1 JTP_APPItfCreate()

This function is the equivalent of the socket call in Linux; it creates a closed connection and returns a descriptor of the created connection.

The input to this function is :

- **lgdp :** this is the global structure of eJTP, where eJTP keeps all its state and configuration.
- **iconn :** this is a pointer to the platform specific *connection* structure. If for example JTP runs within the Linux kernel this should point to the newly created socket. When

JTP runs in PSS/PLF a custom connection structure is used. It is the responsibility of the platform specific part to pass the correct structure or if needed – as in the PSS/PLF case – create the structure.

A.1.2 JTP_AppIntfSetConnOpt()

This function is used by the application to communicate with eJTP and the ATC module in order to set specific parameters. If an option is not recognized by eJTP the set option command is forwarded to the ATC module in use. In order for the application to set parameters of the ATC module, it should first use this function to choose which ATC module to use. If no ATC module is chosen by the application a default one is used.

The input to this function is :

- **lgdp :** this is the global structure of eJTP, where eJTP keeps all its state and configuration.
- **conn :** this is a pointer to the platform independent connection structure. The platform dependent part is responsible for receiving either the connection id (the case in PSS/PLF) or the platform dependent connection structure (in case of Linux kernel) and retrieve the pointer to this structure.
- **optionId :** the id of the option to be set. Current valid options are :
  - **JTP_PKTZ_MOD :** To set the ATC module.
  - **JTP_IN_ORDER_DELIVERY :** To set whether the ADUs should delivered in order or not. This option is available in the default and in the voice ATC modules.
  - **JTP_MAX_DELAY:** To set the maximum delay for an ADU. The value is in milliseconds. A value of 0 signifies infinite delay, while a value of $0xFFFF$ signifies that the ATC module should set this value. This option is available only in the voice ATC.

  - **JTP_LOSS_TOLERANCE:** To set the Loss Tolerance of the application. The values is between 0 and 100, where 0 signifies reliable transfer and 100 is equivalent to UDP. This option is available in the default and in the voice ATC modules.

- **value** : a pointer to a buffer from which the new value of the option is fetched.

- **optSize** : the size of the buffer pointed to by value

A.1.3 JTP_AppIntfGetConnOpt()

This function is used by the application to interact with eJTP and the ATC module in order to get the value of specific parameters. If an option is not recognized by eJTP the get option command is forwarded to the ATC module in use. In order for the application to get parameters of the ATC module, it should first use this function to choose which ATC module to use. If no ATC module is chosen by the application a default one is used.

The input to this function is :

- **lgdp :** this is the global structure of eJTP, where eJTP keeps all its state and configuration.

- **conn :** this is a pointer to the platform independent connection structure. The platform dependent part is responsible for receiving either the connection id (the case in PSS/PLF) or the platform dependent connection structure (in case of Linux kernel) and retrieve the pointer to this structure.

- **optionId :** the id of the option to be retrieved. The options are the same as in the JTP_AppIntfSetConnOpt() call.

- **value** : a pointer to a buffer where the value of the option is saved.

- **optSize** : the size of the buffer pointed to by value

A.1.4 JTP_AppIntfConnect()

This function is used when the application wants to initiate a connection with a remote host. The connection should have already been created using JTP_AppIntfCreate(). Also the platform specific part is responsible for receiving the address from the application and updating the connection structure accordingly. This is left to the platform specific part since the address structure used in different platforms can vary significantly. As mentioned earlier eJTP shared code has only non blocking calls. In most platforms the connect() call is a blocking call that returns when the connection has been established. To achieve this behavior the caller should provide a call back function to be invoked when the connection has been established or if an error has occurred.

The input to this function is :

- **lgdp :** this is the global structure of eJTP, where eJTP keeps all its state and configuration.
- **conn :** this is a pointer to the platform independent connection structure. The platform dependent part is responsible for receiving either the connection id (the case in PSS/PLF) or the platform dependent connection structure (in case of Linux kernel) and retrieve the pointer to this structure.
- **connectCallBackFxnPtr** : this is the call back function provided for JTP to invoke when the connection has been completed. The prototype for this connection is:

  $typedef\ void\ (*JTP\_ConnConnectCallBackFxn)(JTP\_GlobalData\_t * lgdp, void * conn, int\ retCode)$

  Where the input is:

  - **lgdp :** this is the global structure of eJTP, where eJTP keeps all its state and configuration.
  - **conn :** this is a pointer to the platform independent connection structure. The platform dependent part is responsible for receiving either the connection id (the

case in PSS/PLF) or the platform dependent connection structure (in case of Linux kernel) and retrieve the pointer to this structure.

- **retCode :**
  - JTPRet_OK (0), if the connection was successfully established
  - JTPRet_TIMEOUT (negative), if the connection establishment timed out
  - JTPRet_CR (negative), if the remote host rejected the connection
  - JTPRet_ERROR (negative), if another error occurred

A.1.5 JTP_AppIntfBind()

This function is provided only by the platform specific part, since it might be desirable to use the platform's port manager (e.g. Linux kernel version). The functionality of this function in the platform specific part should associate a connection with a specific port where this connection will be receiving data from.

A.1.6 JTP_AppIntfListen()

This function is called when an application wants to listen on a specific port for incoming connections. Before the application makes this call it should first bind to a specific port.

The input to this function :

- **lgdp :** this is the global structure of eJTP, where eJTP keeps all its state and configuration.
- **conn :** this is a pointer to the platform independent connection structure. The platform dependent part is responsible for receiving either the connection id (the case in PSS/PLF) or the platform dependent connection structure (in case of Linux kernel) and retrieve the pointer to this structure.
- **backlog :** this parameter is similar to the backlog argument of the Linux listen() call. For each listening connection eJTP maintains two queues:

- *incompleteQueue* : this queue contains all the connections that have been initiated by remote hosts but have not yet been established.
- *completeQueue* : this queue contains all the established connections with remote hosts that have not yet been accepted by the listening application

Backlog is the maximum sum of the lengths for both queues.

A.1.7 JTP_AppIntfAccept()

This function is called by the application in order to accept an incoming connection from a remote host on a listening port. The application before calling this function should first create a connection, bind to a specific port and invoke a listen for this port. In most platforms the accept() call is a blocking call. If there is no connection already established the function will block until a connection establishes. To achieve this behavior the caller should provide a call back function to be invoked when a connection has been established.

The input to this function is :

- **lgdp :** this is the global structure of eJTP, where eJTP keeps all its state and configuration.
- **conn :** this is a pointer to the platform independent connection structure. The platform dependent part is responsible for receiving either the connection id (the case in PSS/PLF) or the platform dependent connection structure (in case of Linux kernel) and retrieve the pointer to this structure.
- **acceptCallBackFxnPtr** : a pointer to the function that JTP should invoke when a connection is established.

  $typedef\ void\ (*JTP\_ConnAcceptCallBackFxn)$
  $(JTP\_GlobalData\_t\ *lgdp,\ void*oldConn,\ void*newConn,\ void*addr,\ uint16*addrLen)$

  where the input is:

  - **lgdp :** this is the global structure of eJTP, where eJTP keeps all its state and configuration.
  - **oldConn :** a pointer to the **listening** platform independent connection structure. The platform dependent part is responsible for finding the connection id and return the new connection to the correct application.
  - **newConn :** a pointer to the **new** platform independent connection structure. The platform dependent part is responsible for returning to the application only the connection id of the new connection.
  - **addr :** a pointer to the buffer that was provided by the application in the JTP_AppInfAccept() call (see below), where the address of the remote host should be saved.
  - **addrLen :** a pointer to the buffer containing the length of the address structure, like the previous argument, this buffer has been provided by the application.

- **addr :** a pointer to a buffer allocated by the application to store address of the remote host, the platform specific part is responsible for setting the correct address.
- **addrLen :** a pointer to a buffer allocated by the application that contains the allocated space for the addr buffer.

A.1.8 JTP_AppIntfSend()

This function is called by the application in order to send an Application Data Unit (ADU) to the destination. If JTP fails to accept the whole ADU, i.e. JTP ran out of memory this function will return an error.

The input to this function is :

- **lgdp :** this is the global structure of eJTP, where eJTP keeps all its state and configuration.

- **conn :** this is a pointer to the platform independent connection structure. The platform dependent part is responsible for receiving either the connection id (the case in PSS/PLF) or the platform dependent connection structure (in case of Linux kernel) and retrieve the pointer to this structure.
- **sendBuff** : A pointer to the buffer containing the ADU.
- **sendBuffLen** : The length of the buffer containing the ADU.

A.1.9 JTP_AppIntfRecv()

This function is invoked by the application in order to receive an ADU If this function is invoked for a connection that is not established then the behavior is similar to the receive-from() function in linux. In this case the caller must provide a buffer to store the address of the node that the ADU was received from. In most platforms the receive() call is a blocking call that returns when the data has been received. To achieve this behavior the caller should provide a call back function to be invoked when an ADU is ready to be delivered to the application or if an error has occurred.

The input to this function is :

- **lgdp :** this is the global structure of eJTP, where eJTP keeps all its state and configuration.
- **conn :** this is a pointer to the platform independent connection structure. The platform dependent part is responsible for receiving either the connection id (the case in PSS/PLF) or the platform dependent connection structure (in case of linux kernel) and retrieve the pointer to this structure.
- **rcvBuff** : A pointer to the buffer to store the received ADU.
- **rcvBuffLen** : The length of the buffer to store the received ADU.
- **addr :** a pointer to a buffer allocated by the application to store address of the remote host, the platform specific part is responsible for setting the correct address.

- **addrLen :** a pointer to a buffer allocated by the application that contains the allocated space for the addr buffer.

- **recvCallBackFxnPtr** : a pointer to the function that JTP should invoke when an ADU is ready to be delivered to the application.

  $typedef\ void\ (*JTP\_ConnRcvCallBackFxn)$
  $(JTP\_GlobalData\_t\ *lgdp,\ void\ *conn,\ \ void\ *buff,\ uint16\ *\ buffLen)$

  where the input is:

  - **lgdp :** this is the global structure of eJTP, where eJTP keeps all its state and configuration.
  - **conn :** a pointer to the connection structure that received this ADU.
  - **buff :** a pointer in the buffer containing the recvd adu.
  - **buffLen :** a pointer to a buffer containing the actual length of the received ADU.

A.1.10 JTP_AppIntfClose()

This function is called by the application in order to close a connection. In most platforms close() is a blocking call that returns only when the connection is closed. To achieve this behavior the caller should provide a callback function to be invoked when the connection has closed or if an error has occurred.

The input to this function is :

- **lgdp :** this is the global structure of eJTP, where eJTP keeps all its state and configuration.

- **conn :** this is a pointer to the platform independent connection structure. The platform dependent part is responsible for receiving either the connection id (the case in PSS/PLF) or the platform dependent connection structure (in case of linux kernel) and retrieve the pointer to this structure.

- **closeCallBackFxnPtr** : a pointer to the function that JTP should invoke when the connection has closed.

  $typedef\ void$
  $(*JTP\_ConnCloseCallBackFxn)(JTP\_GlobalData\_t\ *lgdp,\ TLA\_AppId\_t\ appId)$

  where the input is:

  - **lgdp :** this is the global structure of eJTP, where eJTP keeps all its state and configuration.
  - **appId :** the connection id of the closed connection.

A.2 ATC API

JTP design supports the concurrent operation of multiple Application Transfer Controller modules in order to give to the applications the capability to choose the appropriate ATC based on their requirements. To facilitate the implementation of new ATC modules, JTP defined a set of functions that an ATC should implement in order to be incorporated with the rest of JTP.

A.2.1 Initialize and Destroy functions

All JTP modules have to implement a set of functions that are executed when the system starts up or shuts down. This functions are responsible for initializing any necessary state and ensuring that a clean shut-down happens.

void* JTPAtc_Init()

This function initializes any necessary state that the ATC module needs to maintain.

void JTPAtc_Exit(void *state)

This function should make sure that the ATC module has a clean exit. It shouldn't release the passed buffer, since JTP is going to release this memory after the return of this function.

void JTPAtc_Free(void* state)

This function should release all memory that the ATC module has allocated, except from

the **state** buffer, JTP will release this function.

A.2.2 Functions for interacting with the application Interface

The ATC module needs to interact with the application interface, since the application interface is the module that enables the communication with the application. Each ATC should implement the following functions:

JTPRetCode_t JTPAtc_Connect(void *state, JTP_Conn_t *conn, void* data, int datalen)

JTPRetCode_t JTPAtc_SetConnOption(void *state, JTP_Conn_t *conn, int optionId, void *value, int optSize)

JTPRetCode_t JTPAtc_GetConnOption(void *state, JTP_Conn_t *conn, int optionId, void *value, int *optSize)

JTPRetCode_t JTPAtc_Close(JTP_GlobalData_t *lgdp, JTP_Conn_t *conn)

JTPRetCode_t JTPAtc_Recv(void *state,JTP_Conn_t *conn, uint8* recvBuff, uint16 *recvBuffLenPtr)

JTPRetCode_t JTPAtc_Send(void *state,JTP_Conn_t *conn, uint8* aduBuff, uint16 aduBuffLen)

void JTPAdu_FreeIn(void* adu) void JTPAdu_FreeOut(void* adu) JTPRetCode_t JTPAtc_GetRawnd(void *state, JTP_Conn_t *conn, uint16 *rawn)

JTPRetCode_t JTPAtc_GetSackFing

(void *state, JTP_Conn_t *conn, JTP_PacketID_t *firstPack,

JTP_PacketID_t *lastPack, JTP_SackFng sack)

JTPRetCode_t JTPAtc_AddDataToPkt(void *state, TLA_Packet_t *pkt, uint16 dataLength)

JTPRetCode_t JTPAtc_ProcessPkt(void *state, TLA_Packet_t *pkt)

JTPRetCode_t JTPAtc_ProcessConnRequest(void *state, TLA_Packet_t *pkt)

JTPRetCode_t JTPAtc_ProcessTimeout(void *state,int type,TLA_AppId_t appId)

Appendix B

Packet Formats

In this section a description for the JTP packet is provided. The packet headers are presented in figure B

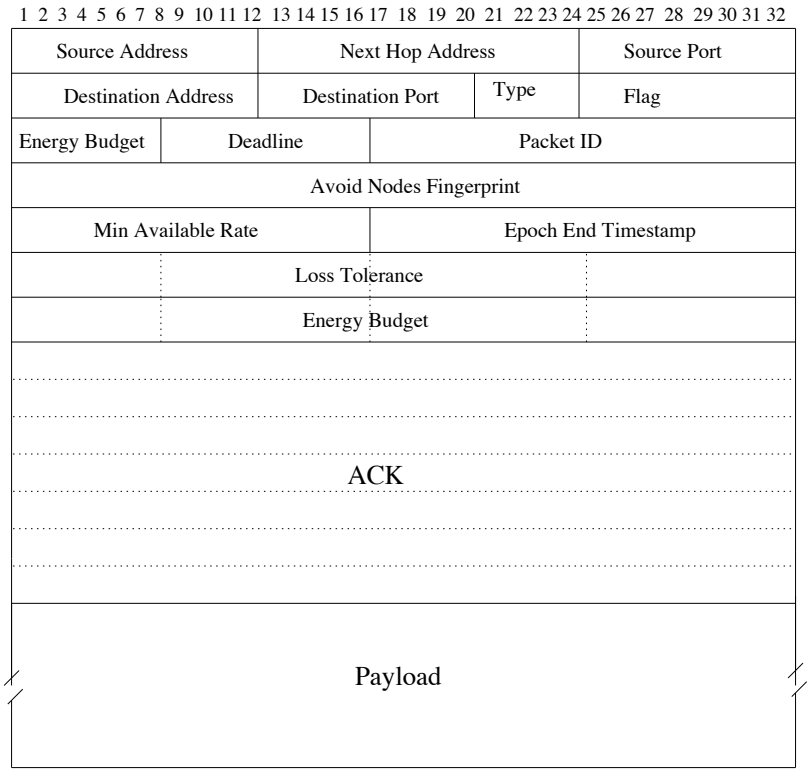


Figure B.1: The JTP packet

A small description of all the fields in the header:

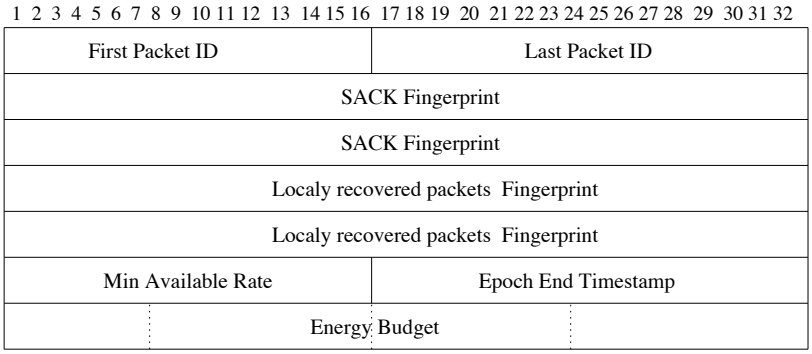


Figure B.2: The ACK portion of the JTP packet

- **source address bits 1-12** The address of the sender.
- **next hop address bits 13-24** In JTP the end to end transfer occurs with point to point transmissions. That means that for the forwarding module is packet is destined to the next hop along the path.
- **source port bits 25-32** The port where the application at the sender listen for data.
- **destination address bits 33-44** The address of the receiver
- **destination port bits 45-52** The port where the application at the receiver listen for data.
- **Type bits 52-56** This field describe the type of the packet. The available types are
  - **0000**:Reset packet
  - **0001**:Connection Establishment Packet
  - **0010**:Connection Rejection Packet
  - **0011**:Data packet
  - **0100**:Selective Acknowledgment Packet
  - **0101**:Negative ACKnowledgement no-route. This packet is generated by mid-path nodes when a node has no route to the final destination.

- **Flags bits 57-64** This field contains flags as a way to mark packets. For now the only flags used are
  - **bit 57** : This packet will request from the receiver to send a SACK.
  - **bit 58** : This bit is set to 1 when a packet is rerouted. If a packet is rerouted then the receiver
  - **bit 59** : This bit is set to 1 when the sender does not require any acknowledgment from the receiver.
  - **bit 60**: This bit is set if the packet is the last packet of a connection.
- **Energy Budget bits 65-72** In this field we assign the energy budget for the packet.
- **Deadline bits 73-80** In this field we should assign a deadline for the packet.
- **Packet ID bits 80-96** In case this packet also carries a data packet,this field contains its ID.
- **Avoid Nodes bits 97-128** If a packet gets rerouted this field contains the ids of the nodes that should be avoided when the new route is calculated.
- **Minimum available rate bits 129-138** This field is initialized to all ones at the sender. Every node updates this field so in the end it will contain the minimum available rate in the path.
- **Stability time stamp bits 139-154** This time stamp indicates until when this path will be stable. The sender initializes this field to all ones. Every node updates the field so as in the end it will contain the time until when this path is expected to be stable.
- **Energy budget bits 155-160** In this field each node is going to add the necessary energy in order to transmit a packet over this link with the requested reliability.

- **ACK Information bits 161-320** If this packet is an acknowledgment packet sent by the receiver then in these 20 packets the receiver will include the information that should be packet then this 20 bytes contain the feedback information. Figure B shows in detail the information contained in the ACK portion of the packet.
  - **SACK fingerprint bits 161-224** In this field the receiver provides the IDs of the packet that he would like to be retransmitted.
  - **Mid-path nodes retransmitted packets fingerprint 245-288** This field contains the IDs of the packet that the receiver requested for a retransmission but some mid-path node has retransmitted.
  - **Minimum available rate bits 289-296**
  - **Stability time stamp bits 397-312**
  - **Energy budget bits 313-320** These three fields contain the pat information that the receiver got from the last data packet that wasn't rerouted.
- **Data bits 321-** The rest of the packet contains the payload of the packet. If this packet is not a SACK then the data segment begins at the bit 161.

Bibliography

[AACS87] A. Aggarwal, B. Alpern, A. Chandra, and M. Snir. A model for hierarchical memory. In *Proceedings of the nineteenth annual Association for Computing Machinery (ACM) symposium on Theory of Computing (STOC)*, STOC '87, pages 305–314, New York, NY, USA, 1987. Association for Computing Machinery (ACM).

[AHH05] J. Ahn, S. Hong, and J. Heidemann. An adaptive FEC code control algorithm for mobile wireless sensor networks. *Journal of Communications and Networks*, 7(4):489–499, December 2005.

[AHM03] L. Andrew, S. Hanly, and R. Mukhtar. CLAMP: A System to Enhance the Performance of Wireless Access Networks. In *Proceedings of Global Telecommunications (GLOBECOM)*, 2003.

[AJ03] E. Altman and T. Jimenez. Novel delayed ack techniques for improving tcp performance in multihop wireless networks. In *Personal Wireless Communications*, 2003.

[All09] M. Allman. Comments on selecting ephemeral ports. *Special Interest Group on Data Communication (SIGCOMM) Computer Communications Review*, 39(2):13–19, March 2009.

[APS99] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control, 1999. Request For Comment (RFC) 2581.

[AS04] V. Anantharaman and R. Sivakumar. TCP Performance Over Mobile Ad-hoc Networks - A Quantitative Study. *Wireless Communications and Mobile Computing*, 4:203–222, 2004.

[ASA00] A. Aggarwal, S. Savage, and T. Anderson. Understanding the Performance of TCP Pacing. In *Proceedings of INFOCOM*, pages 1157–1165, 2000.

[ASM+11] B. Anand, J. Sebastian, Soh Yu Ming, A.L. Ananda, Mun Choon Chan, and R.K. Balan. Pgtp: Power aware game transport protocol for multi-player mobile games. In *Communications and Signal Processing (ICCSP), 2011 International Conference on*, pages 399 –404, feb. 2011.

[ASSC02] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38, 2002.

[BA03] H. Bai and M. Atiquzzaman. Error modeling schemes for fading channels in wireless communications: A survey. *Communications Surveys Tutorials, Institute of Electrical and Electronic Engineers (IEEE)*, 5(2):2 –9, 2003.

[Bar89] R. Barden. Requirements for Internet Hosts – Communication Layers, October 1989. Request For Comment (RFC) 1122.

[BB95] A. Bakre and B. R. Badrinath. I-TCP: Indirect TCP for Mobile Hosts. In *Proceedings of International Conference on Distributed Computing Systems (ICDCS)*, 1995.

[BG92] D.P. Bertsekas and R.G. Gallager. *Data networks*. Prentice-Hall international editions. Prentice Hall, 1992.

[BHJ10] A. Bemporad, M. Heemels, and M. Johansson. *Networked Control Systems*. Springer, 2010.

[BK98] H. Balakrishnan and R. Katz. Explicit Loss Notification and Wireless Web Performance. In *Institute of Electrical and Electronic Engineers (IEEE)*

Global Telecommunications (GLOBECOM) Global Internet, Sydney, Australia, November 1998.

[BM93] L. Benmohamed and S.M. Meerkov. Feedback control of congestion in packet switching networks: the case of a single congested node. *Institute of Electrical and Electronic Engineers (IEEE)/Association for Computing Machinery (ACM) Transactions on Networking*, 1(6):693–708, December 1993.

[BM02] D. Barman and I. Matta. Effectiveness of Loss Labeling in Improving TCP Performance in Wired/Wireless Networks. In *Proceedings of International Conference on Network Protocols (ICNP)'2002: The 10th Institute of Electrical and Electronic Engineers (IEEE) International Conference on Network Protocols*, Paris, France, November 2002.

[BMAA04] D. Barman, I. Matta, E. Altman, and R. El Azouzi. TCP Optimization through FEC, ARQ and Transmission Power Tradeoffs. In *Proceedings of WWIC 2004: The 2nd International Conference on Wired/Wireless Internet Communications*, Frankfurt (Oder), Germany, February 2004.

[BN00] I. Batsiolas and I. Nikolaidis. Selective idling: An experiment in transport layer power-efficient protocol implementation. In *International Conference on Internet Computing*, pages 419–426, 2000.

[BOP94] L.S. Brakmo, S.W. O'Malley, and L.L. Peterson. Tcp vegas: New techniques for congestion detection and avoidance. In *Proceedings of the Conference on Communications Architectures, Protocols and Applications*, Special Interest Group on Data Communication (SIGCOMM) '94, pages 24–35, New York, NY, USA, 1994. Association for Computing Machinery (ACM).

[BPSK97] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz. A Comparison of Mechanisms for Improving TCP Performance Over Wireless links.

Institute of Electrical and Electronic Engineers (IEEE)/Association for Computing Machinery (ACM) Transactions on Networking, 5(6):756–769, 1997.

[BRBR03a] D. Bertozzi, A. Raghunathan, L. Benini, and S. Ravi. Transport Protocol Optimization for Energy Efficient Wireless Embedded Systems. In *Proceedings of Design, Automation and Test in Europe (DATE)*, 2003.

[BRBR03b] D. Bertozzi, A. Ranghunathan, L. Benini, and S. Ravi. Transport Protocol Optimization For Energy Efficient Wireless Embedded. *DATE: Design Automation and Test in Europe Conference and Exhibition*, 2003.

[BSAK95] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz. Improving TCP Performance Over Wireless Networks. In *Proceedings of Association for Computing Machinery (ACM) MobiCom*, 1995.

[BV99] S. Biaz and N. Vaidya. Discriminating Congestion Losses From Wireless Losses Using Inter-Arrival Times At The Receiver. In *Proceedings of Institute of Electrical and Electronic Engineers (IEEE) Symposium ASSET*, 1999.

[BYJK07] G. Balakrishnan, M. Yang, Y. Jiang, and Y. Kim. Performance analysis of error control codes for wireless sensor networks. In *Information Technology, 2007. ITNG '07. Fourth International Conference on*, pages 876 –879, april 2007.

[CC84] R. Comroe and D. Costello. ARQ Schemes for Data Transmission in Mobile Radio Systems. *Selected Areas in Communications, Institute of Electrical and Electronic Engineers (IEEE) Journal on*, 2(4):472–481, 1984.

[CDN+96] A. Chankhunthod, P.B. Danzig, C. Neerdaels, M.F. Schwartz, and K.J. Worrell. A hierarchical internet object cache. In *Proceedings of the 1996 Annual Conference on USENIX Annual Technical Conference*, ATEC '96, pages 13–13, Berkeley, CA, USA, 1996. USENIX Association.

[CGM+01] C. Casetti, M. Gerla, S. Mascolo, M. Sanadidi, and R. Wang. TCP Westwood: Bandwidth Estimation for Enhanced Transport Over Wireless Networks. In *Proceedings of Association for Computing Machinery (ACM) MobiCom*, 2001.

[Che88] D. R. Cheriton. Vmtp: Versatile message transaction protocol, 1988.

[Chi05] M. Chiang. Balancing transport and physical layers in wireless multihop networks: jointly optimal congestion control and power control. *Selected Areas in Communications, Institute of Electrical and Electronic Engineers (IEEE) Journal on*, 23(1):104–116, 2005.

[CI95] R. Caceres and L. Iftode. Improving the performance of reliable transport protocols in mobile computing environments. *Selected Areas in Communications, Institute of Electrical and Electronic Engineers (IEEE) Journal on*, 13(5):850 –857, jun 1995.

[CJ89] D. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and Integrated Services Digital Network (ISDN) Systems*, 17(1):1–14, June 1989.

[CJ03] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol, 2003. Request For Comment (RFC) 3626.

[CK74] V. Cerf and R. Kahn. A protocol for packet network intercommunication. *Communications, Institute of Electrical and Electronic Engineers (IEEE) Transactions on*, 22(5):637 – 648, may 1974.

[Cla88] D. Clark. The design philosophy of the darpa internet protocols. In *Symposium proceedings on Communications architectures and protocols*, Special Interest Group on Data Communication (SIGCOMM) '88, pages 106–114, New York, NY, USA, 1988. Association for Computing Machinery (ACM).

[CLC+08] A. Chan, S. Lee, X. Cheng, S. Banerjee, and P. Mohapatra. The impact of link-layer retransmissions on video streaming in wireless mesh networks. In *WICON '08: Proceedings of the 4th Annual International Conference on Wireless Internet*, pages 1–9, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[CLGS06] J. Chen, Y.Z. Lee, M. Gerla, and M.Y. Sanadidi. Tcp with delayed ack for wireless networks. In *Broadband Communications, Networks and Systems, 2006. BROADNETS 2006. 3rd International Conference on*, 2006.

[CLZ87] D. D. Clark, M. L. Lambert, and L. Zhang. NETBLT: a High Throughput Transport Protocol. In *Special Interest Group on Data Communication (SIGCOMM) '87: Proceedings of the Association for Computing Machinery (ACM) Workshop on Frontiers in Computer Communications Technology*, pages 353–359, New York, NY, USA, 1987. Association for Computing Machinery (ACM) Press.

[CNV04] K. Chen, K. Nahrstedt, and N. Vaidya. The Utility of Explicit Rate-Based Flow Control in Mobile Ad Hoc Networks. In *Proceedings of WCNC*, 2004.

[CRVP98] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash. A Feedback Based Scheme for Improving TCP Performance in Ad-Hoc Wireless Networks. In *Proceedings of International Conference on Distributed Computing Systems (ICDCS)*, 1998.

[CSW03] Y. Chen, E. Sirer, and S.B. Wicker. On selection of optimal transmission power for ad hoc networks. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03)*, HICSS '03, pages 300.3–, Washington, DC, USA, 2003. Institute of Electrical and Electronic Engineers (IEEE) Computer Society.

[CT90a] D. Clark and D. Tennenhouse. Architectural Considerations for a New Generation of Protocols. In *Proceedings of Association for Computing Machinery (ACM) Special Interest Group on Data Communication (SIGCOMM)*, pages 200–208, 1990.

[CT90b] D. Clark and D. L. Tennenhouse. Architectural Considerations for a New Generation of Protocols. In *Proceedings of Association for Computing Machinery (ACM) Special Interest Group on Data Communication (SIGCOMM)*, 1990.

[CV02] S. Chandra and A. Vahdat. Application-specific Network Management for Energy-aware Streaming of Popular Multimedia Formats. In *Proceedings of USENIX*, 2002.

[CWT01] H. Che, Z. Wang, and Y. Tung. Analysis and design of hierarchical web caching systems. In *INFOCOM 2001. Twentieth Annual Joint Conference of the Institute of Electrical and Electronic Engineers (IEEE) Computer and Communications Societies. Proceedings. Institute of Electrical and Electronic Engineers (IEEE)*, volume 3, pages 1416 –1424 vol.3, 2001.

[DDC87] Lixia Zhang David D. Clard, Mark L. Lambert. NETBLT: A Bult Data Transfer Protocol, 1987. Request For Comment (RFC) 998.

[DMCOC93] A. DeSimone, C. Mooi Choo, and Y. On-Ching. Throughput performance of transport-layer protocols over wireless lans. In *Global Telecommunications Conference, 1993, including a Communications Theory Mini-Conference. Technical Program Conference Record, Institute of Electrical and Electronic Engineers (IEEE) in Houston. Global Telecommunications (GLOBECOM) '93., Institute of Electrical and Electronic Engineers (IEEE)*, pages 542–549 vol.1, Nov-2 Dec 1993.

[ea00] R. Stewart et al. Stream Control Transmission Protocol, 2000. Request For Comment (RFC) 2960.

[EBW02] J. Ebert, B. Burns, and A. Wolisz. A trace-based approach for determining the energy consumption of a wlan network interface, 2002.

[EKCD00] T.A. ElBatt, S.V. Krishnamurthy, D. Connors, and S. Dao. Power management for throughput enhancement in wireless ad-hoc networks. In *Communications, 2000. International Conference on Communications (ICC) 2000. 2000 Institute of Electrical and Electronic Engineers (IEEE) International Conference on*, volume 3, pages 1506–1513 vol.3, 2000.

[FML02] Z. Fu, X. Meng, and S. Lu. How Bad TCP Can Perform in Mobile Ad Hoc Networks. In *Proceedings of ISCC*, 2002.

[FN01] L.M. Feeney and M. Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *INFOCOM 2001. Twentieth Annual Joint Conference of the Institute of Electrical and Electronic Engineers (IEEE) Computer and Communications Societies. Proceedings. Institute of Electrical and Electronic Engineers (IEEE)*, volume 3, pages 1548 –1557 vol.3, 2001.

[For07] B. Ford. Structured streams: a new transport abstraction. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, Special Interest Group on Data Communication (SIGCOMM) '07, pages 361–372, New York, NY, USA, 2007. Association for Computing Machinery (ACM).

[Fox89] R. Fox. TCP Big Window and Nak Options, 1989. Request For Comment (RFC) 1106.

[FPEN05] G.F. Franklin, J.D. Powell, and A. Emami-Naeini. *Feedback Control of Dynamic Systems*. Prentice Hall, 5 edition, 2005.

[GML02] Z. F. Greenstein, B. X. Meng, and S. Lu. Design and Implementation of a TCP-friendly Transport Protocol for Ad Hoc Wireless Networks. In *Pro-*

ceedings of the 10th Institute of Electrical and Electronic Engineers (IEEE) International Conference on Network Protocols, 2002.

[Gol05] A. Goldsmith. *Wireless Communications*. Cambridge University Press, 2005.

[GSK94] R. Goel, M. Singla, and K.Bansal. On Importance of Bit Error Rate in Wireless Communications. *Ubiquitous Computing and Communication Journal*, 2(3), 1994.

[GW02] A. Goldsmith and S. Wicker. Design challenges for energy-constrained ad hoc wireless networks. *Wireless Communications, Institute of Electrical and Electronic Engineers (IEEE) [see also Institute of Electrical and Electronic Engineers (IEEE) Personal Communications]*, 9(4):8–27, 2002.

[GWMC09] D. Gupta, D. Wu, P. Mohapatra, and C. Chuah. Experimental comparison of bandwidth estimation tools for wireless mesh networks. In *INFOCOM 2009, Institute of Electrical and Electronic Engineers (IEEE)*, pages 2891 –2895, april 2009.

[HA97] Z. J. Haas and P. Agrawal. Mobile-TCP: An Asymmetric Transport Protocol Design for Mobile Systems. In *Proceedings of International Conference on Communications (ICC)*, 1997.

[Hav99] P.J.M. Havinga. Energy efficiency of error correction on wireless systems. In *Wireless Communications and Networking Conference (WCNC), 1999. Institute of Electrical and Electronic Engineers (IEEE)*, pages 616 –620 vol.2, 1999.

[HDPT04] J.L. Hellerstein, Y. Diao, S. Parekh, and D.M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[HFGN12] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida. The newreno modification to tcp's fast recovery algorithm, 2012.

[HV02] G. Holland and N. Vaidya. Analysis of TCP Performance Over Mobile Ad Hoc Networks. *Wireless Networks*, 8:275–288, 2002.

[HVB01] G. Holland, N. Vaidya, and P. Bahl. A rate-adaptive mac protocol for multi-hop wireless networks. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, MobiCom '01, pages 236–251, New York, NY, USA, 2001. Association for Computing Machinery (ACM).

[HWW04] C. Hoene, S. Wieth?lter, and A. Wolisz. Predicting the Perceptual Service Quality Using a Trace of VoIP Packets. In *Proceedings of the Fifth International Workshop on Quality of Future Internet Services (QofIS'04)*, Barcelona, Spain, September 2004.

[int] Integrity. http://www.ghs.com/products/rtos/integrity.html.

[(IT96] International Telecommunication Union (ITU). Interface between Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DCE) for terminals operating in the packet mode and connected to public data networks by dedicated circuit, 1996. ITU-T Recommendation X.25.

[Jac88] V. Jacobson. Congestion avoidance and control. *Special Interest Group on Data Communication (SIGCOMM) Computer Communication Review*, 18(4):314–329, August 1988.

[Jai86] R. Jain. A timeout-based congestion control scheme for window flow-controlled networks. *Selected Areas in Communications, Institute of Electrical and Electronic Engineers (IEEE) Journal on*, 4(7):1162 – 1167, oct 1986.

[Jia02] C. Jiao. *Wireless Data Transmission: Error Characterization and Performance Analysis*. PhD thesis, Wayne State University, 2002.

[JLMV02] P. Jacquet, A. Laouiti, P. Minet, and L. Viennot. Performance of multipoint relaying in ad hoc mobile routing protocols. In *Proceedings of the Second*

International IFIP-TC6 Networking Conference on Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; and Mobile and Wireless Communications, NETWORKING '02, pages 387–398, London, UK, 2002. Springer-Verlag.

[JM96] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, pages 153–181. Kluwer Academic Publishers, 1996.

[KHF06] E. Kohler, M. Handley, and S. Floyd. Designing dccp: congestion control without reliability. In *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, Special Interest Group on Data Communication (SIGCOMM) '06, pages 27–38, New York, NY, USA, 2006. Association for Computing Machinery (ACM).

[KHR02] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. In *Special Interest Group on Data Communication (SIGCOMM) '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 89–102, 2002.

[KK00] R. Kravets and P. Krishnan. Application-driven Power Management for Mobile Communication. *Wireless Networks*, 6:263–277, 2000.

[KR97] A. Kolarov and G. Ramamurthy. A control theoretic approach to the design of closed loop rate based flow control for high speed atm networks. In *Proceedings of the INFOCOM '97. Sixteenth Annual Joint Conference of the Institute of Electrical and Electronic Engineers (IEEE) Computer and Communications Societies*, INFOCOM '97, pages 293–, Washington, DC, USA, 1997. Institute of Electrical and Electronic Engineers (IEEE) Computer Society.

[KR99] A. Kolarov and G. Ramamurthy. A control-theoretic approach to the design of an explicit rate controller for abr service. *Institute of Electrical and Electronic Engineers (IEEE)/Association for Computing Machinery (ACM) Transactions on Networking*, 7(5):741–753, October 1999.

[KRH+06] S. Katti, H. Rahul, D. Hu, W. Katabi, M. Médard, and J. Crowcroft. Xors in the air: practical wireless network coding. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Special Interest Group on Data Communication (SIGCOMM) '06, pages 243–254. Association for Computing Machinery (ACM), 2006.

[KSE+04] R. Krishnan, J.P.G. Sterbenz, W.M. Eddy, C. Partridge, and M. Allman. Explicit Transport Error Notification (ETEN) for Error-prone Wireless and Satellite Networks. *Computer Networks*, 46(3):343–362, 2004.

[LCF+10] W. Li, E. Chan, G. Feng, D. Chen, and S. Lu. Analysis and performance study for coordinated hierarchical cache placement strategies. *Computer Communications*, 33(15):1834 – 1842, 2010.

[LCL07] H. Lee, A. Cerpa, and P. Levis. Improving wireless simulation through noise modeling. In *Information Processing in Sensor Networks (IPSN), 2007. 6th International Symposium on*, 2007.

[LCS06] N. Laoutaris, H. Che, and I. Stavrakakis. The lcd interconnection of lru caches and its analysis. *Performance Evaluation*, 63(7):609–634, July 2006.

[LDJ04] Y. Liaw, A. Dadej, and A. Jayasuriya. Estimating Throughput Available to a Node in Wireless Ad-hoc Network. In *Proceedings of Mobile Adhoc and Sensor Systems (MASS) 2004*, pages 555–557, Fort Lauderdale, Florida, 2004.

[LHY+06] H.K. Lee, V. Hall, K.H. Yum, K.I. Kim, and E.J. Kim. Bandwidth estimation in wireless lans for multimedia streaming services. In *Multimedia and Expo,*

2006 Institute of Electrical and Electronic Engineers (IEEE) International Conference on, pages 1181 –1184, july 2006.

[lin] Linux. http://www.linux.org/.

[LPP04] K. Lakshminarayanan, V.N. Padmanabhan, and J. Padhye. Bandwidth estimation in broadband access networks. In *Proceedings of the 4th Association for Computing Machinery (ACM) Special Interest Group on Data Communication (SIGCOMM) conference on Internet measurement*, IMC '04, pages 314–321, New York, NY, USA, 2004. Association for Computing Machinery (ACM).

[LS01] J. Liu and S. Singh. ATCP: TCP for Mobile Ad Hoc Networks. *Institute of Electrical and Electronic Engineers (IEEE) Journal on Selected Areas in Communications*, 19(7):1300–1315, 2001.

[LSLLG11] R. Laufer, T. Salonidis, H. Lundgren, and P. Le Guyadec. Xpress: a cross-layer backpressure architecture for wireless multi-hop networks. In *Proceedings of the 17th annual international conference on Mobile computing and networking*, MobiCom '11, pages 49–60, New York, NY, USA, 2011. Association for Computing Machinery (ACM).

[Lud00a] R. Ludwig. Eliminating Inefficient Cross-layer Interactions in Wireless Networking, April 2000. Doctoral Dissertation, Technischen Hochshule Aachen.

[Lud00b] R.E. Ludwig. *Eliminating Inefficient Cross-Layer Interactions in Wireless Networking.* PhD thesis, Technischen Hochschule Aachen, April 2000.

[MBNP06] S. Mao, D. Bushmitch, S. Narayanan, and S.S. Panwar. Mrtp: a multiflow real-time transport protocol for ad hoc networks. *Institute of Electrical and Electronic Engineers (IEEE) Transactions on Multimedia*, 8(2):356–369, 2006.

[MCG96] S. Mascolo, D. Cavendish, and M. Gerla. Atm rate based congestion control using a smith predictor: an eprca implementation. In *INFOCOM '96. Fifteenth Annual Joint Conference of the Institute of Electrical and Electronic Engineers (IEEE) Computer Societies. Networking the Next Generation. Proceedings Institute of Electrical and Electronic Engineers (IEEE)*, volume 2, pages 569 –576 vol.2, mar 1996.

[mic] Mica2 mote datasheet. www.xbow.com.

[MK05] Prasant Mohapatra and SrikanthV. Krishnamurthy, editors. *Ad Hoc Networks*. Springer US, 2005.

[MK10] P. Mohapatra and S. Krishnamurthy. *AD HOC NETWORKS: Technologies and Protocols*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[ML11] F. Gont M. Larsen. Recommendations for Transport-Protocol Port Randomization, 2011. Request For Comment (RFC) 6056.

[MMFR96] M. Mathis, J. Madhavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options, 1996. Request For Comment (RFC) 2018.

[Mon05] D.C. Montgomery. *"Introduction to Statistical Quality Control"*. John Wiley & Sons, New York, 5th edition edition, 2005.

[Moo05] T.K. Moon. *Error Correction Coding*. John Wiley & Sons, 2005.

[MTK02] A. Markopoulou, F. Tobagi, and M. Karam. Assessment of VoIP Quality over Internet Backbones. In *Proceedings of Institute of Electrical and Electronic Engineers (IEEE) Infocom*, June 2002.

[NG10] V. Namboodiri and L. Gao. Energy-efficient voip over wireless lans. *Institute of Electrical and Electronic Engineers (IEEE) Transactions on Mobile Computing*, 9(4):566 –581, april 2010.

[OOBB04] Ruy De Oliveira, Ruy De Oliveira, Torsten Braun, and Torsten Braun. A dynamic adaptive acknowledgment strategy for tcp over multihop wireless networks. In *In Proceedings of Institute of Electrical and Electronic Engineers (IEEE) INFOCOM*, 2004.

[opn] OPNET. http://www.opnet.com/.

[oSC81] Information Sciences Institute University of Southern California. Transmission Control Protocol, 1981. Request For Comment (RFC) 793.

[PAP+95] S. Paul, E. Ayanoglu, T.F. La Porta, K.-W.H. Chen, K.E. Sabnani, and R.D. Gitlin. An asymmetric protocol for digital cellular communications. *Institute of Electrical and Electronic Engineers (IEEE) Computer and Communications Societies, Annual Joint Conference of the*, 3:1053, 1995.

[PCP12] I. Psaras, W.K. Chai, and G. Pavlou. Probabilistic in-network caching for information-centric networks. In *Proceedings of the second edition of the Information-Centric Networking (ICN) workshop*, ICN '12, pages 55–60, New York, NY, USA, 2012. Association for Computing Machinery (ACM).

[Per01] C.E. Perkins. *Ad hoc networking.* Addison-Wesley networking : mobile computing. Addison-Wesley, 2001.

[PFTK98] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *Proceedings of Association for Computing Machinery (ACM) Special Interest Group on Data Communication (SIGCOMM)*, Vancouver, British Columbia, September 1998.

[PLS+06] J.-S. Park, D.S. Lum, F. Soldo, M. Gerla, and M. Medard. Performance of network coding in ad hoc networks. In *Military Communications Conference, 2006. MILCOM 2006. Institute of Electrical and Electronic Engineers (IEEE)*, pages 1 –6, oct. 2006.

[PW72] W.W. Peterson and E.J. Weldon. *Error-correcting codes*. MIT Press, 1972.

[RBS00] S. Raman, H. Balakrishnan, and M. Srinivasan. ITP: An Image Transport Protocol for the Internet. In *Proceedings of International Conference on Network Protocols,*, 2000.

[RCP+04] J. Redi, I. Castineyra, C. Partridge, K. Manning, R. Rosales-Hain, R. Ramanathan, and S. Kolek. Joint Architecture Vision for Low Energy Networking (JAVeLEN)—DARPA-ATO Connectionless Networking Program, Phase I. Technical report, BBN Technical Report 8408, December 2004.

[RFB99] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP, 1999. Request For Comment (RFC) 3168.

[RGGP06] S. Rangwala, R. Gummadi, R. Govindan, and K. Psounis. Interference-aware fair rate control in wireless sensor networks. *Special Interest Group on Data Communication (SIGCOMM) Computer Communication Review*, 36(4):63–74, August 2006.

[RK01] R. Rozovsky and P. R. Kumar. SEEDEX: A MAC Protocol for Ad Hoc Networks. In *Proceedings of the 2nd Association for Computing Machinery (ACM) International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2001.

[RKM+08] J. Redi, S. Kolek, K. Manning, C. Partridge, R. Rosales-Hain, R. Ramanathan, and I. Castineyra. JAVeLEN—An Ultra-Low Energy Ad Hoc Wireless Network. *Ad Hoc Networks*, 6(1):108–126, 2008.

[RKT10] E.J. Rosensweig, J. Kurose, and D. Towsley. Approximate models for general cache networks. In *INFOCOM, 2010 Proceedings Institute of Electrical and Electronic Engineers (IEEE)*, pages 1–9, 2010.

[RM98] K. Ratnam and I. Matta. WTCP: An Efficient Mechanism for Improving TCP Performance Over Wireless Links. In *Proceedings of Third Institute of Electrical and Electronic Engineers (IEEE) Symposium on Computers and Communications*, 1998.

[SAHS03] K. Sundaresan, V. Anantharaman, H-Y. Hsieh, and R. Sivakumar. ATP: A Reliable Transport Protocol for Ad-hoc Networks. In *Proceedings of Association for Computing Machinery (ACM) MobiHoc*, Annapolis, ML, 2003.

[Sal07] D. Salomon. *Data Compression: The Complete Reference*. Springer-Verlag, 2007.

[SBS02] E. Shih, P. Bahl, and M. J. Sinclair. Wake on Wireless: An Event-driven Energy Saving Strategy for Battery Operated Devices. In *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2002.

[SCFJ96] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications, 1996. Request For Comment (RFC) 1889.

[SFR03] W.R. Stevens, B. Fenner, and A.M. Rudoff. *Unix Network Programming: The Sockets Networking Api, Volume 1*. Prentice Hall, 2003.

[SH03] F. Stann and J. Heidemann. RMST: Reliable Data Transport in Sensor Networks. In *Proceedings of the First International Workshop on Sensor Net Protocols and Applications*, 2003.

[Shi07] R. Shirey. Internet Secirity Glossary, Version 2, 2007. Request For Comment (RFC) 4949.

[SKK03] J. Strauss, D. Katabi, and F. Kaashoek. A measurement study of available bandwidth estimation tools. In *Proceedings of the 3rd Association for Comput-*

ing Machinery (ACM) Special Interest Group on Data Communication (SIGCOMM) conference on Internet measurement, Internet Measurement Conference (IMC) '03, pages 39–44, New York, NY, USA, 2003. Association for Computing Machinery (ACM).

[SKK07] D. Silva, F.R. Kschischang, and R. Koetter. A rank-metric approach to error control in random network coding. In *Information Theory for Wireless Networks, 2007 Institute of Electrical and Electronic Engineers (IEEE) Information Theory Workshop on*, pages 1 –5, july 2007.

[SKSK02] B. Sadeghi, V. Kanodia, A. Sabharwal, and E. Knightly. Opportunistic media access for multirate ad hoc networks. In *Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 24–35. Association for Computing Machinery (ACM), 2002.

[SR98] S. Singh and C. S. Raghavendra. PAMAS—power aware multi-access protocol with signalling for ad hoc networks. *Special Interest Group on Data Communication (SIGCOMM) Computer Communication Review*, 28(3):5–26, July 1998.

[SR02] R. C. Shah and J. M. Rabaey. Energy aware routing for low energy ad hoc sensor networks. *Wireless Communications and Networking Conference (WCNC), 2002. Institute of Electrical and Electronic Engineers (IEEE)*, 1:350–355 vol.1, 2002.

[SRC81] J.H. Saltzer, D.P. Reed, and David Clark. End-to-End Arguments in System Design. In *Proceedings of the Second International Conference on Distributed Computing Systems*, pages 509–512, August 1981.

[SRS01] C. Santivanez, R. Ramanathan, and I. Stavrakakis. Making Link-State Routing Scale for Ad Hoc Networks. In *Proceedings of the Association for Com-*

puting Machinery (ACM) International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), pages 22–32, 2001.

[Ste02] C. Steinbach. *A Reinforcement-Learning Approach to Power Management*. PhD thesis, MIT, 2002.

[SVSB99] P. Sinha, N. Venkitaraman, R. Sivakumar, and V. Bharghavan. WTCP: A Reliable Transport Protocol for Wireless Networks. In *Proceedings of Association for Computing Machinery (ACM) MobiCom*, 1999.

[SZ99] I. Stoica and H. Zhang. Providing Guaranteed Services Without Per Flow management. In *Proceedings of Association for Computing Machinery (ACM) Special Interest Group on Data Communication (SIGCOMM)*, 1999.

[Tan02] A. Tanenbaum. Implementing an Operating System, April 2002.

[TBV99] V. Tsaoussidis, H. Badr, and R. Verma. Wave wait protocol (wwp): an energy-saving transport protocol for mobile ip-devices. In *Network Protocols, 1999. (International Conference on Network Protocols (ICNP) '99) Proceedings. Seventh International Conference on*, pages 301 – 308, oct.-3 nov. 1999.

[TE92] L. Tassiulas and A. Ephremides. Stability properties of constrained queuing systems and scheduling policies for maximum throughput in multihop radio networks. *Automatic Control, Institute of Electrical and Electronic Engineers (IEEE) Transactions on*, 37(12):1936 –1948, dec 1992.

[TM] Inc. The MathWorks. MATLAB. http://www.mathworks.com/.

[Tor] K.H. Torvmark. Low Power Systems Using the CC1010. Chipcon Application Note AN017.

[VHS84] D. Velten, R. Hinden, and J. Sax. Reliable data protocol, 1984.

[Wat89] R.W. Watson. The delta-t transport protocol: features and experience. In *Local Computer Networks, 1989., Proceedings 14th Conference on*, pages 399–407, 1989.

[WCK02] C. Wan, A. T. Campbell, and L. Krishnamurthy. PSFQ: A Reliable Transport Protocol for Wireless Sensor Networks. In *Proceedings of Wireless sensor networks and applications (WSNA)*, 2002.

[Wea92] A.C. Weaver. The Xpress Transfer Protocol. In *Proceedings of International Workshop on Advanced Communications and Applications for High Speed Networks*, 1992.

[WFC09] Q. Wang, A. Feng, and J. Cao. Available bandwidth estimation in ieee 802.11 ad hoc networks. In *Hybrid Intelligent Systems (HIS), 2009. Ninth International Conference on*, volume 1, pages 135 –137, aug. 2009.

[WJHR09] A. Warrier, S. Janakiraman, Sangtae Ha, and I. Rhee. Diffq: Practical differential backlog congestion control for wireless networks. In *INFOCOM 2009, Institute of Electrical and Electronic Engineers (IEEE)*, pages 262 –270, april 2009.

[WLS+07] C. Wang, B. Li, K. Sohraby, M. Daneshmand, and Y. Hu. Upstream congestion control in wireless sensor networks through cross-layer optimization. *Selected Areas in Communications, Institute of Electrical and Electronic Engineers (IEEE) Journal on*, 25(4):786 –795, may 2007.

[YF04] O. Younis and S. Fahmy. Heed: A hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks. *Institute of Electrical and Electronic Engineers (IEEE) Transactions on Mobile Computing*, 3:366–379, 2004.

[YGE01] Y. Yu, R. Govindan, and D. Estrin. Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks. Technical report, UCLA, 2001.

[Yod99] V. Yodaiken. The rtlinux manifesto. In *In Proceedings of The 5th Linux Expo*, 1999.

[YR95] C. Yang and A.V.S. Reddy. A taxonomy for congestion control algorithms in packet switching networks. *Network, Institute of Electrical and Electronic Engineers (IEEE)*, 9(4):34 –45, jul/aug 1995.

[YZJ04] H. Yousefizadeh, L. Zheng, and H. Jafarkhani. Rate Constrained Power Control in Space-Time Coded Fading Ad-Hoc Networks. In *Proceedings of Institute of Electrical and Electronic Engineers (IEEE) Global Communications Conference (GLOBECOM) 2004)*, November 2004.

[ZBP01] W. Zhang, M.S. B., and S.M. Phillips. Stability of networked control systems. *Control Systems, Institute of Electrical and Electronic Engineers (IEEE)*, 21(1):84 –99, feb 2001.

[ZCF05] H. Zhai, X. Chen, and Y. Fang. Rate-Based Transport Control for Mobile Ad Hoc Networks. In *Proceedings of Wireless Communications and Networking Conference (WCNC)*, 2005.

Curriculum Vitae

Personal

Niky Riga

Raytheon BBN Technologies

10 Moulton Street

Cambridge MA, 02138, USA +1 617 873-3160(office)

+1 617 873-6091(fax)

nriga@bbn.com

Research Interests

- Research Testbeds: build, use and federation of research testbeds.
- Future Internet Architectures.
- Transport Protocols and energy conservation techniques in Wireless Sensor and Ad-Hoc Networks.
- Disruption Tolerant Networks, and Content Based Routing techniques, especially in the DTN setting.

Education

- **Boston University,** USA 2013, PhD in Computer Science
- **Boston University** USA 2007 Master of Arts in Computer Science.
- **National Technical University of Athens,** Greece 2001 Diploma in Electrical and Computer Engineering.

Work Experience

- **Network Scientist,** Raytheon BBN Technologies (September 2010 – now).
  Network Scientist in Raytheon BBN Technologies. As a Network Scientist I have been a member of the GENI Project Office, that is a nation-wide virtual laboratory for network and distributed systems research.
- **Staff Scientist,** Raytheon BBN Technologies (February 2007 – September 2010).
  Staff Scientist in Raytheon BBN Technologies. During my employment, I have worked on multiple projects focused mainly on designing, evaluating and deploying novel network protocols for Ad-Hoc Wireless Networks. Since March 2010, I have been working on the GENI project.
- **Research Assistant,** Boston University (January 2004 – January 2007).
  Worked on various projects on Wireless Ad-Hoc Networks (under NSF grant). My research is primarily focused on designing and testing transport protocol schemes for Wireless Ad-Hoc Networks. I am working on devising cross layer techniques that will optimize network wise energy expenditure.
- **Intern,** Raytheon BBN Technologies (June 2005 – April 2006).
  Worked on the JAVeLEN (Joint Architecture Vision for Low Energy Networking) project. My work was focused on designing, specifying and implementing the transport protocol for the JAVeLEN system.

- **Research Assistant,** Boston University (Summer 2003).

  Worked with Prof. G. Kollios and T.M. Murali on Dimensionality Reduction Techniques.

- **Teaching Assistant,** Boston University (2002 – 2003).

  Introduction to Computer Science (CS101), Introduction to Web Computing (CS103).
  I organized and taught the lab section*s of these courses. I also designed and graded the midterm and final exam for the lab section*s.

- **Network Administrator,** National Technical University of Athens (2000 – 2002).

  Network and System administrator for the Network Operating Center. Responsible for the web broadcasting of the Greek National Radio and Television. Responsible for configuring, managing and supporting the University's teleconferencing and teleteaching facilities. Responsible for providing specifications, support and consulting for developing similar facilities in other academic institutions.

- **Teaching Assistant,** National Technical University of Athens (2000 – 2002).

  Introduction to Computer Programming.
  I taught basic programming skills using Pascal in the lab section* of the course.
  Programming Techniques.
  I taught basic programming techniques using C in the lab section* of the course.

- **Database Administrator,** Archaeological Institute of Crete (AIC) (Summer 1999).

  I was the Database Administrator for AIC. I was also responsible for collecting topological information about Archaeological sites and design maps using the ArcView system.

- **Instructor,** Computer Explorers Institute (1998 – 2000).

  I was the instructor for introductory Computer classes for children and adults.

Service

- **Conducted tutorials at ICDCS, TridentCom, SIGCSE, NSDI**
- **Member of the organizing committee for 1st and 2nd GENI Summer camps**
- **Member of the Program Committee for GENI Research and Education Experiment Workshops**
- **Live Broadcast Coordinator**, 13th IEEE International Conference on Network Protocols (ICNP), November 6-9, 2005 Boston, Massachusetts.
- **Network Reading Group Coordinator**, Boston University (2004 – 2005).
  I was responsible for organizing the weekly seminars of the Networking group.
- **Welcoming Committee Member, Poster Session Organizer**, 4th Workshop on Applications and Services in Wireless Networks (ASWN), August 8-11, 2004, Boston Massachusetts.
- **Welcoming Committee Member**, 15th International Conference on Scientific and Statistical Database Management (SSDBM), July 9-11, 2003, Cambridge Massachusetts.
- **Reviewer for various top tier conferences and journals**,

Projects

- **Global Environment for Network Innovations** GENI is an NSF funded program which aims in developing and deploying a unique virtual laboratory for at-scale networking experimentation where the researchers can envision, create, test and deploy new technologies for the future internets. Within this project, my focus is to help with the successful integration and execution of a rich variety of experiments that make meaningful use of GENI capabilities.

- **Content Based Access** Within the PIRANA and SPINDLE III projects, we designed and evaluated a routing algorithm that supports content-based access. A content router differs from a normal router in that it routes packets based on the contained information and not based on the node's ids.

- **Joint Architecture Vision for Low Energy Networking** The goal of this program is to reduce the energy-per-bit used for delivering information in a wireless ad hoc network by 100 times, while still delivering the delay and throughput capabilities of traditional wireless multihop networks. Within this project, I worked in designing, evaluating and implementing JTP, a Transport Protocol that would significantly reduce energy consumption, without compromising the application's performance.

- **Interactive Voice Response System** I designed and implemented an IVR system for the Administrative Departments of National Technical University of Athens. I used the Envox Development platform, and Dialogic analog and ISDN cards.

- **Multimedia Integration System** I designed and developed an automated system for synchronizing multimedia objects with the slides of a presentation. The system creates a SMIL file that manages all the multimedia objects and has an interactive interface with the user. The output of the system can be viewed by any player that supports SMIL technology. I implemented the system in PHP. This system is used for e-learning purposes at National Technical University of Athens.

Conference Publications

- **Slinky: An Adaptive Protocol for Content Access in Disruption-Tolerant Ad Hoc Networks** Vikas Kawadia, Niky Riga, Jeff Opper and Dhananjay Sampath *International Workshop on Tactical Mobile Ad Hoc Networking, held in conjunction with ACM SIGMOBILE MobiHoc 2011* **Rated Best of Workshop** Paris, France May 2011

- **An Energy-conscious Transport Protocol for Multi-hop Wireless Networks.** N. Riga, I. Matta, A. Medina,C. Partridge, J. Redi *CoNEXT,* , New York, NY December 2007.

- **Transport Protocols for Energy Constrained Environments.** N. Riga, A. Medina, I. Matta, C. Partridge, J. Redi, I. Castineyra. *WIP Session. ACM SIGCOMM*, Philadelphia, PA. August 2005.

- **DIP: Density Inference Protocol for wireless sensor networks and its application to density-unbiased statistics,** N. Riga, I. Matta, A. Bestavros, *Proc. of the Second International Workshop on Sensor and Actor Network Protocols and Applications (SANPA)*, Boston, United States, August 2004.

- **Providing Soft Bandwidth Guarantees Using Elastic TCP-based Tunnels,** M. Guirguis, A. Bestavros, I. Matta, N. Riga, G. Diamant, Y. Zhang, *In proceedings of the 9th IEEE Symposium on Computer and Communications (ISCC)*, Alexandria, Egypt, July 2004.

Posters

- **Contour Map and Event Boundary Detection in Sensor Networks,** N. Riga, I. Matta, A. Bestavros, *At the Science and Engineering Day of Boston University*, Boston, United States, March 2005.

- **Density Inference in Wireless Sensor Networks,** N. Riga, I. Matta, A. Bestavros, *At the 4th Workshop on Applications and Services in Wireless Network (ASWN)*, Boston, United States, August 2004.

- **MBG: Minimum Bandwidth Guarantees,** M. Guirguis, A. Bestavros, I. Matta, N. Riga, G. Diamant, Y. Zhang, *At the IAP Research Day at Boston University*, Boston, United States, April 2004.

Skills

- **Excellent knowledge of:**
- C, C++ (and STL), Java, Python, Perl
- SQL, XML, HTML, DHTML, PHP, JavaScript.
- Unix network programming,
- MySQL, SQL Server, NS, Matlab
- OpenFlow, Emulab, PlanetLab
- **Had some experience with:**
- OPNET, TinyOS
- Linux kernel programming and module implementation.