COOPERATIVE CONGESTION CONTROL IN POWER GRID COMMUNICATION NETWORKS

BY

NAVEEN CHERUKURI

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Adviser:

Prof. Klara Nahrstedt

ABSTRACT

Power Grid Digital Communication networks are needed for the delivery of Phasor Measurement Unit (PMU)[1] sensor data with real time guarantees to the control centers for timely decisions. Present day network systems are unable to ensure the real-time needs of the PMU data as they are not designed to support PMU devices. Hence, the need arises for new communication and control protocols. This thesis describes the Cooperative Congestion Control framework to ensure real-time guarantees for PMU data and to respond if transient deviations in real-time PMU network traffic occur. The framework utilizes a) NASPI[3], [4] aligned multiple service class queuing architecture; b) Cooperative real-time flow scheduling and Bandwidth reassignment; c) Cooperative coordination and back-pressure among neighboring nodes. It then yields real-time PMU data guarantees during transient traffic pattern changes and/or overload situations. Our experiments confirm that the framework delivers real-time performance very well under different transient scenarios.

To my parents, friends and colleagues for their love and support.

ACKNOWLEDGMENTS

I would like to thank my adviser Prof. Klara Nahrstedt for all of the advice she has given me along the way. Tim Yardley helped me with the interdisciplinary aspects of power grid, providing valuable suggestions and access to the testbed. Erich Heine and Raoul Rivas contributed constructive suggestions and I am thankful for them. I also wish to thank all the administrative members of TCIPG for their support. This material is based upon work supported by the Department of Energy under Award Number DE-OE0000097.

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Overview

Power grid status monitoring and control capabilities are very significant to understand and manage complex power generation and transmission patterns, to avoid blackouts etc. The present day power grid digital communication system, i.e, Supervisory Control and Data Access (SCADA) is in use from about 40 years. With the introduction of Phasor Measurement Unit (PMU) sensors, the need for more flexible and adaptive communication networks became inevitable. The SCADA communication system features a centralized star-topology, point to point communication, severe bandwidth constraints and proprietary protocols which are not sufficient to meet the requirements of todays grid. Limitations of SCADA are discussed in more detail in [7].

Infrastructure critical networks such as power grid digital communication networks are those that have very strict requirements on real-time availability, reliability and security. These networks need to assure end-to-end latency and throughput guarantees to the control and data packets of many critical flows.

Intelligent decision making at the control centers require wide-area situational awareness about the grid, which provides comprehensive view of the entire interconnection when multiple utilities' measurements are combined.

This is made possible by the PMUs which can provide time synchronized and precise grid measurements called synchrophasor measurements.

North American SynchroPhasor Initiative's (NASPI)[2] vision is to improve power system reliability through wide-area measurement, monitoring and control. NASPI is working to develop an industrial grade, secure, standardized, distributed, and expandable data communications infrastructure, called the NASPI network or NASPInet, to support synchrophasor applications that depend on shared PMU data. A conceptual architecture of NASPInet and its functional and security requirements are captured in NASPInet specification documents[3], [5] commissioned by the U.S. DOE. Figure 1.1 shows a high-level conceptual architecture envisioned for

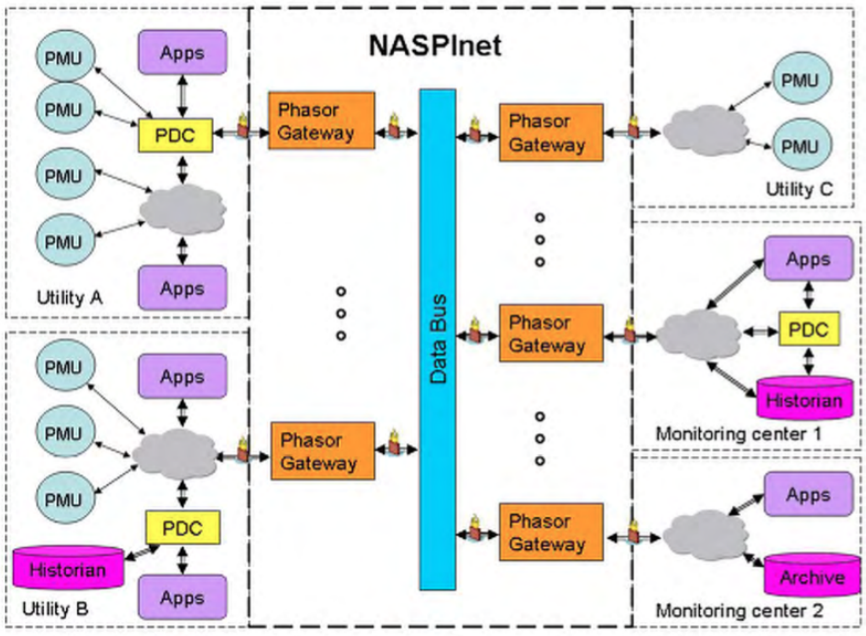

Figure 1.1: NASPInet Architecture

NASPInet. NASPInet is composed of Phasor Gateways (PGWs) and a Data Bus (DB). The DB includes a Wide Area Network (WAN) and associated services to provide basic connectivity, QoS management, performance monitoring, cyber security, and policy enforcement over data exchanged through

NASPInet. Utilities A, B and C are the power grid sensing stations (e.g., substations). Each utility contains PMU sensors, Intelligent Electronic Devices (IEDs) and other applications (Apps) aggregated at the Phasor Data Concentrators (PDCs) connected to the WAN via the PGW. There are monitoring centers which require the sensors' data for timely decisions.

Designing a deployable NASPInet architecture is comprised of many challenges.

1. Design of distributed WAN for PMU data delivery to the monitoring centers.

2. Meet Quality of Service (QoS) requirements of PMU applications, that have very stringent latency requirements, over a large scale network.

Wide Area Network of the NASPInet can use options ranging from the public Internet, MPLS circuits to utility fiber networks. There are several middleware systems that can provide messaging over such wide area communication networks even when they use heterogeneous underlying networking technologies. Examples of such middleware are, GridStat[7] Data Distribution Service for Real Time Systems (DDS2), System of Systems Common Operating Environment (SOSCOE3), and work by Schantz et. al. [6]. Among these, Gridstat has been developed in the context of the power grid[7].

GridStat is designed to address the need for a flexible and robust communication system in the electrical power grid, and provides a specialization of the publisher-subscriber paradigm. Grid- Stat middleware manages network resources, enables reliable delivery of data to any point and provides QoS (Quality of Service) for data streams. GridStat is divided into two planes; the management plane and the data plane. The management plane consists of a hierarchy of QoS brokers which collectively manage resources and sub-

scriptions in the data plane. The data plane is a virtual message bus and lets publishers provide data to the network and enables subscribers to establish subscriptions to status data through a status router network. The use of QoS, on a per-subscription basis, allows subscribers to specify multiple redundant delivery paths (spatial redundancy), subscription interval and delay.

1.2 Problem Description and Contributions

These infrastructure critical networks (NASPInet) have challenges which the Internet does not possess. Critical data loss (e.g., PMU control data) must be minimal and PMU traffic packets have strict end-to-end deadlines. Since power grid status monitoring is also moving towards Internet protocols and services, the existing Internet(IP) technologies must be modified with new protocols to fit into the real-time PMU requirements. One of the major problems for the power grid critical networks (NASPInet) is the congestion of the PMU data streams (flows) in case of increased traffic demands and/or other transient traffic stress situations. During the period of congestion, the real time flows may not meet their delay requirements leading the NASPInet into an unstable and untrusted state. Hence, early notification of congestion and cooperative congestion control are crucial techniques to respond to unpredictable grid/traffic events and protect real-time PMU flows from violations of their deadlines. In this paper we introduce the *Cooperative Congestion Control (CCC) framework* for handling unpredictable traffic events and changes which cause transient congestion situations in the NASPInet.

"Cooperation" is defined in two ways. One being nodes utilizing the early congestion notification, and the other being nodes using the policy-based correlation between different flows i.e., cooperative flows. Cooperative Con-

gestion Control assumes that all PMU sensing nodes and their flows from each power grid substation cooperate with each other during a transient network change and the CCC system ensures real-time guarantees of critical PMU flows.

The Concept of Cooperative Congestion Control is explored in the wireless scenario[8] where nodes cooperate with each other to acquire access to the medium, but in the wired scenario, considering the large size of the networks and multiple domains, cooperation is explored only in a limited fashion. In the infrastructure critical networks, we certainly have smaller size networks, compared to the Internet, and have more control over the network with service-level agreements from the provider. Hence, nodes can cooperate with each other to eliminate congestion without the fear of overhead. Moreover, in the Internet fairness is more important, while in the power grid priority is more important at the expense of fairness. TCP[10] does not react fast enough to congestion and does not protect real-time flows in terms of their delay requirements when congestion occurs. Hence, TCP is unacceptable in infrastructure critical networks, and its retransmissions and acknowledgements will even worsen the delay situation. Moreover, it doesn't consider the correlation of real-time power grid PMU measurement flows. Hence, there is a need for protocols that provide congestion control and protect real-time guarantees of critical PMU flows. In our solution, we provide congestion control over UDP with low overhead and the correlation of flows taken into account.

Our CCC framework does (1) react fast to unpredictable NASPInet traffic events, which cause transient congestion, (2) protect delay guarantees of real-time PMU flows, and (3) utilize the cooperative and correlated nature of PMU flows coming from the same substation.

1.3 Outline

Chapter 2 provides some background information about powergrid. In chapter 3 we look at some of the related work. In chapter 4 we define the model, problem and set of assumptions considered. In chapter 5 we describe our Cooperative Congestion Control approach and design issues and in chapter 6 implementation is discussed. Chapter 7 evaluates the test results and we conclude in chapter 8.

CHAPTER 2

BACKGROUND

2.1 NASPI

The industry is moving towards wide-area measurement, monitoring and control to improve the reliability of the power grid while meeting the increased power demand. NASPI's[2] mission is to create a robust, widely available and secure synchronized data measurement infrastructure with associated monitoring and analysis tools for better planning and reliable operation of the power grid. NASPI plans for deployment of hundreds of thousands of Phasor Measurement Units (PMUs) across the grid that send data at 30 to 120 samples/second to hundreds of applications. PMUs are sensors that can read current and voltage phasors at a substation bus on the transmission power network. These PMUs give direct access to the state of the grid at any given instant in contrast to having to estimate the state as is done today. NASPI applications have varying requirements classified into four classes based on their data requirements. Typically, feedback control applications like transient stability control fall into Class A, open loop control applications like state estimation fall into Class B, situational awareness applications like visualization and monitoring fall into Class C, post event analysis applications like disturbance analysis fall into Class D and other kinds of research application fall into Class E.

2.2 NASPInet

Sharing PMU data widely, i.e., with other utilities, provides wide area situational awareness. NASPI is working to design a wide area network infrastructure, dubbed NASPInet[3], to enable wide area sharing of PMU data. NASPInet will be composed of Phasor Gateways (PGWs) and a Data Bus (DB). The DB includes a Wide Area Network (WAN) and associated services to provide basic connectivity, QoS management, performance monitoring, and cyber security and policy enforcement over data exchanged through NASPInet. PGW is the sole access point of entities like utilities and monitoring centers (i.e.RCs) to the DB. The PGW will manage the connected devices on the entitys side, manage QoS, administer cyber security and access rights, perform necessary data conversions and interface the entitys own network with the DB. NASPInet is intended to facilitate the secure exchange of both real-time streaming data and historical data. PGWs are expected to support both one-to-one unicast data sharing and one-to-many publisher-subscriber based data sharing in an efficient manner.

2.3 GridStat

GridStat[7], as shown in Figure 4.3, is a publisher-subscriber framework designed to be deployed in the NASPInet Data Bus. GridStat is divided into management and data planes. The management plane consists of QoS broker modules that collectively control and manage resources in the data plane. The data plane is populated by status routers, publishers and subscribers, where publishers provide data and subscribers can subscribe to data. The management hierarchy handles subscription requests and establishes paths from the publisher to the subscriber through a sequence of status routers.

CHAPTER 3

RELATED WORK

Significant amount of research has been done on cooperative congestion control in Vehicular Adhoc Networks (VANET)[8]. The basic idea of applicative-layer congestion control approach is to define policies, in order to dynamically and cooperatively schedule messages transmission in the network. Messages scheduling is carried out according to priorities, evaluated as a function of the utility of the concerned messages, the sender application and the neighborhood context. The messages transmission in the vehicular network is carried out in an efficient and cooperative manner, by favoring vehicles holding the highest-priority messages to send.

In the wired scenario, especially in the internet, congestion control is dealt in a simple manner in the view of the large number of nodes. TCP end-to-end congestion control mechanisms[10] are sufficient for the internet flows which are mostly best effort. For the Premium service flows, service level agreements are done and the overlay protocols react to the congestion notified through the feedback from the receiver. Various congestion control and avoidance techniques in the internet scenario are based on Explicit Congestion Notification (ECN), Random Early Detection (RED) etc.[15], [11], [19].

Asynchronous Transfer Mode (ATM) network is a high-speed network with its link layer protocol that ensures QoS of the data packets. Furthermore, in case of traffic congestion, ATM networks implement back pressure

algorithms[18] and send tokens towards senders to slow down the traffic generation.

Integrated Dynamic Soft Real Time scheduler (iDSRT)[9] is an integrated real-time CPU and network scheduler. It coordinates the prioritization of real time task both in CPU usage phase and network usage phase. In this thesis we are more concerned about the networking phase. *inet* the component responsible for the real-time network scheduling implements queue routines enqueue/deque to support *Earliest Deadline First* (EDF)[12], [13] for the real time packets and $First - in - First - out$ (FIFO) for the best effort packets.

CHAPTER 4

MODELS AND PROBLEM DESCRIPTION

4.1 Power Grid Sensing Application Model

Description	Class	Data Rate in Hz (Samples per second)	Availability in %	Max Interruption in ms	Latency in ms
Feedback Control	A	30,60,120	99.9999%	< 5 ms	< 50 ms
Feed Forward Control	B	20,30,60	99.999%	< 25 ms	< 100 ms
Display	C	10,15,20,30	99.99%	< 100 ms	< 1 s
Disturbance Analysis	D	30,60,120	99.99%	N/A	BE
Research	E	30,60,120	99.99%	N/A	BE

Figure 4.1: Application Classes

The applications are broadly divided into 5 classes in terms of their servicing data criticality[4], [3] as shown in Figure 4.1. PMUs, IEDs and other Apps can be assigned to any class based on the importance. Applications that capture PMU data belong to Class A. On the other hand, applications that capture surveillance video at substations belong to Class C. For simplicity we assume Class A serves High Real Time (HRT), Class B serves Low Real Time (LRT) and Class C serves Best Effort (BE) Apps.

4.2 Device Model

PMU devices produce streams of samples with information about voltage and current signals[1]. The sampling rate is 30 to 60Hz. Previous generation of Power grid sensors, the IED devices, sampled at a lower rate (e.g., 2Hz).

4.2.1 PMU Sensor Data

The synchrophasor standard C37.118 [1] defines the concept of frames for transmitting data from a PMU to a PDC. Basically a Configuration frame, Data frame, Header frame and a Command frame are specified. These have a particular structure and data type associated with them. Configuration frame, Data frame, and Command frame are binary types and Header frame is of ASCII type. The Data frame is the most frequently transmitted message based on the PMU sample rate, and the typical size is of the order of few hundreds of bytes. The variable size in the Data frame is the number of phasors, and analog and digital signals transmitted, depending on the PMU capability. Hence for example, if a serial communication is chosen to transmit the PMU data, the data transfer capability depends on the baud rate of the communication port. The following table shows the typical data transfer capabilities, assuming that the PMU supports up to 12 phasor channels.

Baud rate, (bps)	**Reporting rate (frames / second)**							
	10	**12**	**15**	**20**	**25**	**30**	**50**	**60**
9,600	12	12*	10*	6*	4*	2	0	0
19,200	12	12	12	12	12	10	4*	2
38,400	12	12	12	12	12	12	12	10
56,700	12	12	12	12	12	12	12	12
115,200	12	12	12	12	12	12	12	12

Figure 4.2: Estimated number of phasor channels that can be transmitted at various baud rates and PMU reporting rates over serial port.

4.3 Network Model

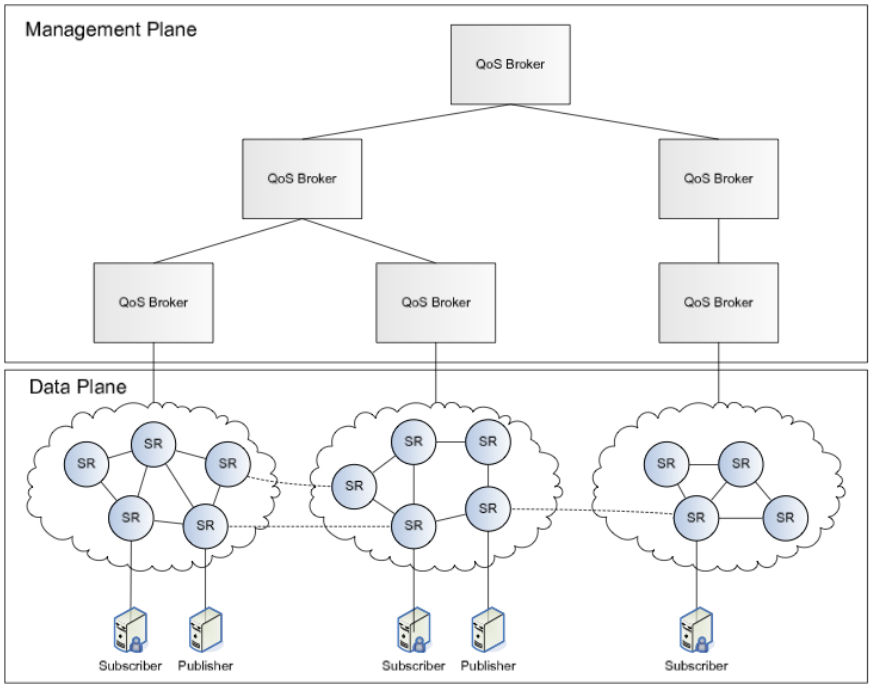

Figure 4.3: Grid-Stat Status Dissemination Architecture

One of the proposed WAN architectures for NASPInet is Grid Stat(Figure 4.3) and we assume a similar management plane model and distributed paradigm (Publish/Subscribe) for our CCC framework. It means, we assume QoS brokers which are responsible for calculating wide area paths and reservations of resources between overlay/status routers (SR). The physical path between the overlay/status routers are then leased with resources, specified by the QoS brokers, to ensure bandwidth, latency and loss rate guarantees at the start of the session.

4.4 Queue Management Model

Each Overlay Router/PGW maintains data queues to serve PMU data streams/flows as well as data from other power grid applications. In our CCC framework we assume applications servicing data belonging to three service classes.

1. Class A : High real time Class with very strict end-end latency requirement, low loss rate etc.
2. Class B : Low real time Class with strict end-end latency requirement, low loss rate etc.
3. Class C : Best Effort Class with no QoS requirements.

Class A of NASPI uses our Class A queue, Class B of NASPI uses our Class B queue and Class C, D, E of NASPI share our Class C queue. Hence, we will assume queuing model consisting of three queues at each PGW, servicing traffic classes A, B, C. Queues are maintained per-class at the Overlay Router/Phasor Gateway.

4.5 Traffic Model

Each flow f in the NASPInet will be characterized by parameters: *minimal sampling rate*, *maximum sampling rate*, *reserved rate*, *peak sample size*, and *average sample size*.

1. *MinRate* - the threshold rate below which the flow should not decrease.
2. *MaxRate* - the ceiling rate above which is considered to be violating the registered condition.
3. *Reserved rate* - the rate reserved along the path of overlay routers by the broker.

4. *Peak Sample Size* - The maximum size of the compressed data packet

5. *Avg. Sample Size* - The average size of the compressed data packet

Each flow originates at the substation, and is generated by sensing devices such as PMUs (each sensing device is called a publisher). Each flow is delivered to a control center that subscribed to receive corresponding PMU sensory data.

4.6 Flow Correlation

Correlation of flows is policy dependent. We consider the *spatial correlation policy* in which different kind of flows (HRT, LRT, BE) that come in from the same substation have certain *amount of redundancy* among them in terms of the information they carry to the control center.

4.7 Problem Description

Real-time guarantees of PMU flows can be violated, when *transient congestion* occurs in the NASPInet. Transient congestion occurs during short term unexpected traffic changes when critical flows do not get sufficient bandwidth, which results in violation of end-to-end delay guarantees and packet drops .

Deviations/congestion of traffic can be caused by:

1. Variable compression of the PMU data or other sensory data (e.g., surveillance video), causing variable bit rate traffic.

2. Increased sending rate of real-time (RT) PMU flows due to unexpected critical event/observation in their sensory space and causing changed

traffic shape of RT flows,

3. Changing demands on real-time traffic requested by control centers (subscribers) due to extended power grid state analysis, causing changes in traffic shapes of RT flows.

For example, in Figure 1.1, if one of the PMU sensors in utility A increases its sampling rate upon observing a critical event, it introduces more traffic into the NASPInet than reserved for it. It is important to stress that current WAN management/data plane related techniques like GridStat do not support fast reaction to temporary, short term, and transient changes in traffic. The current QoS broker-based systems[7] yield QoS guarantees during the stable state, but when congestion happens, global adaptation takes place, i.e., the QoS brokers are contacted and overlay routers wait for QoS broker's response, i.e., until new routing tables/reservations are setup. The time interval between a congestion notification, request for action, and broker response is non-negligible and within this interval transient congestion occurs and real-time PMU flows with their delay requirements are not protected, i.e., violation of deadlines and packet drops due to deadline misses occur.

The qualitative comparison between the system that does only global adaptation(waiting for QoS broker's response) with the system that performs local(Cooperative Congestion Control) and global adaptation is shown in Figure 4.4. We argue that, when only global adaptation is done, during transient congestion state all the class flows(RT and BE) suffer. Suffering can imply missing deadlines, getting dropped etc i.e., the legally received packet rate suffers. Legal packet is the one received within the end-to-end latency bound. But when our local adaptation is used along with the global

adaptation the real time flows never suffer at the expense of best effort flows which are explicitly notified to slow down so as to decrease their dropped number of packets.

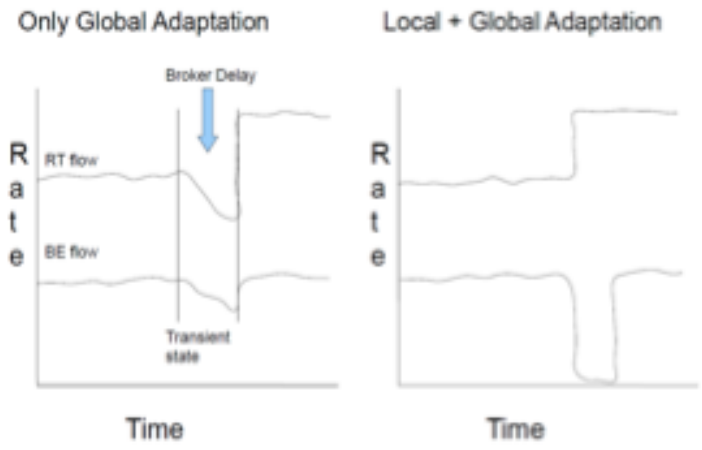

Figure 4.4: Qualitative comparison of global adaptation and local+global adaptation

Hence, we need an approach that will protect real-time PMU flows in NASPInet when transient congestion occurs, and the current reservations cannot handle the increased and changing traffic load.

CHAPTER 5

APPROACH

We propose a Cooperative Congestion Control Framework for the NASPInet data bus to solve the above described problem. The solution is achieved by introducing a) Multiple service class queueing, b) Cooperative real-time flow scheduling, and BW reassignment and c) Cooperative coordination and back pressure approaches among neighboring nodes to counter the transient congestion state. The *multiple service classes* queueing will allow us to differentiate treatment among the service classes A, B, C and use different scheduling policies to respond to congestion situation and protect traffic in real-time classes A and B. *Cooperative real-time scheduling* and *bandwidth reallocation* utilizes the fact that certain flows are cooperative and correlated, containing information about a shared physical space. Hence, the group of the correlated and cooperative real-time flows can help each other, i.e., bandwidth to the LRT and BE flows can be reassigned and lowered to protect the HRT flows at the overlay router/PGW. *Cooperative coordination* allows to expand the protection of RT flows during the congestion to a broader area, informing neighboring overlay router/PGW and/or sources about the experienced congestion and ask them to (a) slow down their traffic towards the congested router, and (b) apply cooperative RT scheduling and BW reallocation at their nodes as well.

Our framework mainly constitutes two components.

1. Designing the overlay router/PGW units that does prioritizing of flows,

real time packet scheduling, bandwidth reassignment adaptation.

2. Congestion notification of the low priority flows to the upstream gateways or overlay routers that are propagated till the particular sender which in turn decreases its rate.

5.1 Overlay Router Design

Cooperative real-time flow scheduling, BW assignment and multiple-service class queueing need differentiating the flows at the overlay router and scheduling real-time flows based on their packet deadlines and priority. The overlay router achieves flow differentiation with three major components as shown in Figure 5.1. The *Execution unit* performs cooperative real time scheduling on multi-service-class queues. The *Adapter unit* executes bandwidth allocation and the congestion notification protocol to achieve cooperative coordination among the neighbors. The *Controller unit* acts as the central component that does all setup activities. The main components in the overlay router are as shown in the figure 5.1 which are.

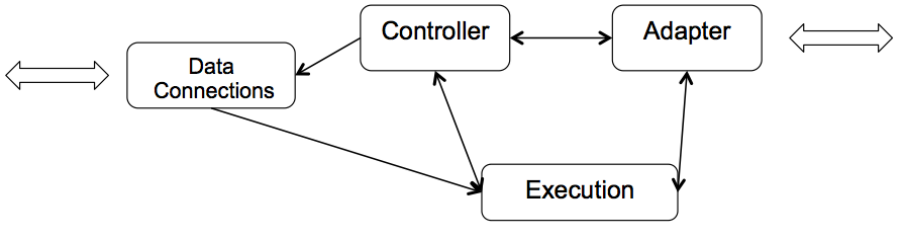

Figure 5.1: Design Components

1. Controller

2. Data Connection Threads

3. Execution

4. Adapter

5.1.1 Controller

This central component is responsible for creating adapter, execution and connections components. It also maintains the metadata, reservation information about the data flows obtained from its QoS broker during the flow set up time. All communication with the QoS broker happens through the controller unit. Metadata includes the information about correlated data flows. For example, in Figure 5.3, flows $f1$, $f2$, $f3$ are the HRT, LRT, BE flows generated from the same substation (i.e., they are correlated). Flows $k1$, $k3$ are HRT, BE flows generated from another substation.

5.1.2 Data Connection Module

This unit handles the connections associated with each publisher-subscriber flow (e.g., *Utility A PMU - Monitoring center 1* as shown in Figure 1.1) at the overlay router.

5.1.3 Execution

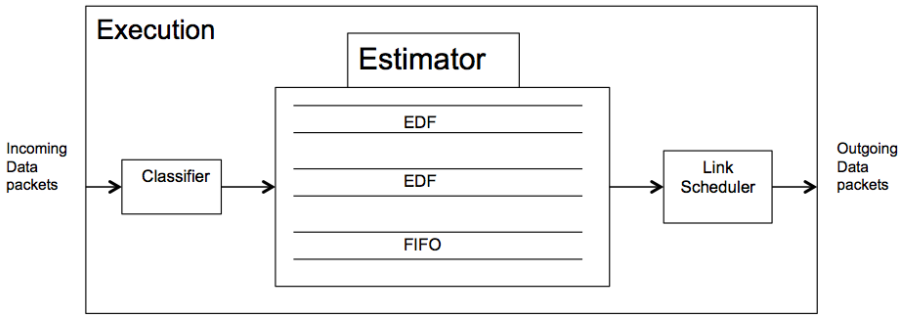

Figure 5.2: Execution Component

Execution unit (Figure 5.2) implements the real-time flow scheduling al-

gorithms and the multiple service class-based queuing. Flows are prioritized depending on their service class (Real time over BE). We introduce the subcomponents of the execution unit as:

1. Packet Classifiers
2. Queueing Strategies
3. Link Scheduler
4. Estimator

Let us describe in detail the subcomponents of the execution unit.

Packet Classifier: We have our own classifier which inspects the incoming packet's *classid/flowid/source address* and places packets into the appropriate service class queue to ensure its QoS requirements (latency, loss rate and throughput). Classifier is pre-notified with the types of flows and the queue corresponding to it. So the classifier simply does a look up into the precomputed table and decided which queue the packets needs to go into.

Queuing Strategy: As described earlier, there are three classes of flows that can exist in the system. Class A and B have queues buffering real time flows. In these classes we need to consider the deadline of packets while dequeuing those queues. We use an $Earliest\ Deadline\ First\ (EDF)$[9] strategy in the enqueue/deque functions for the flows of the Class A and B queues. Class C has no real-time requirements and hence it is a simple $FIFO$ queue.

Link Scheduler: This scheduler is called when the network interface hardware is ready to send the next packet into the network. The outgoing link bandwidth is shared between all classes[14]. Link Scheduler runs a $Weighted\ Round\ Robin$ scheduling policy with weights being the reserved link BW share for each class. For example, Class A queue may get 60% of

link BW, Class B 30% and Class C 10% as shown in Figure 5.3. Some of the link sharing and scheduling algorithms include Class-Based Queuing[14], [20], Hierarchical-Fair Service Curve Algorithm[23], [21]. By allowing isolation between realtime and best effort traffic in cooperation with packet scheduling algorithms that give priority to the real-time traffic, controlled link-sharing is a key component in enabling deployment of priority based packet scheduling algorithms designed to meet the end-to-end service requirements of real time traffic.

Estimator: The estimator determines packet arrival rate and bandwidth usage for each class/flow over a period. Bandwidth usage is estimated by calculating the sent amount of data over a period. Packet arrival rate is calculated using Exponential Weighted Moving Average (EWMA) over the queue size. The two key parameters of the estimator are the instantaneous and history weightage for the estimator and the frequency with which the estimator updates the rate of each class/flow. Let s be the size of the packet in bytes, let b be the link-sharing bandwidth allocated to the class/flow in bytes per second. The algorithm is shown in **Algorithm 1**.

Algorithm 1 Estimation Algorithm

Require: Initialization
  Initialization:
  $period \Leftarrow windowsizeinseconds$
  $w \Leftarrow weight, 0 \leq w \leq 1$
  $s \Leftarrow packetsize$
  $Avg\_rate \Leftarrow b(inbytespersecond)$
  Iteration:
  for each period **do**
    $N \Leftarrow numberofbytessent$
    $M\_rate \Leftarrow N/period$
    $Avg\_rate \Leftarrow (1-w) * Avg\_rate + w * M\_rate$
  end for

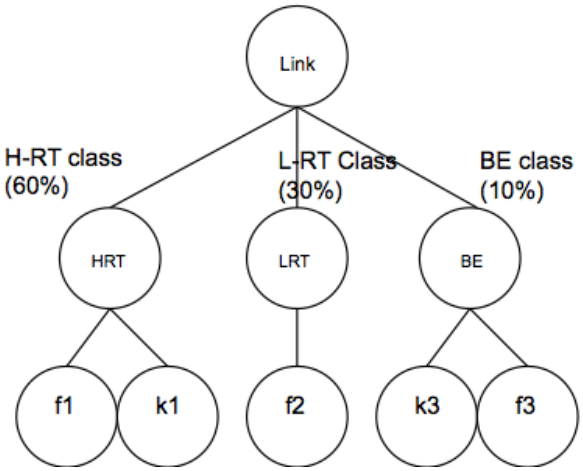

Figure 5.3: Class Hierarchy at the Overlay Router

5.1.4 Adapter

Adapter is the primary component where the *BW reassignment* algorithm and *Cooperative coordination* protocol reside. Its main responsibilities include

1. Maintain the link sharing queue hierarchy and update statistics periodically

2. Adapt to any transient congestion occurrence by bandwidth reassignment

3. Communicate with the adapters of other nodes notifying them about the new rate (BW) assignments.

Class hierarchy Structure: Adapter maintains a class hierarchy structure(Figure 5.3) corresponding to each outgoing link. This structure does the book-keeping of bandwidth shares, usage, drops, backlog, demand, priority etc. per each flow and per each class (e.g., for *f1, f2, f3, k1, k3* flows and their HRT, LRT, BE classes membership).

1. *Bandwidth shares* - Flow-wise and Class-wise reserved bandwidth.

2. *Usage* - Flow-wise and Class-wise bandwidth usage.

3. *Demand* - Flow-wise and Class-wise bandwidth demand.

4. *Priorities* - High Real time - 0, Low Real time -1, Best Effort - 2.

In the above Figure 5.3, the link-sharing class hierarchy structure specifies the desired policy in terms of the division of bandwidth for a particular link. For example, the link is shared by a number of high real-time, low-real time and non-real-time traffic classes and flows. The high real time flows $f1$ and $k1$ are represented as the leaf nodes under the parent High real time class. In this, the class HRT flows contains delay-sensitive PMU real time traffic. Similarly the class LRT also has flows that are delay sensitive and real time but slightly less requirements than the HRT class flows. The BE class flows are the best effort flows with no guarantees.

1. Each class/flow is characterized by

   (a) *assured rate* AR (the reserved rate).

   (b) *minimum rate* MR(the minimum threshold rate of the class/flow).

   (c) *ceil rate* CR (above which the rate cannot increase)

   (d) *priority* P (HRT>LRT>BE and the flows of the same class have the same priority)

   (e) *actual rate* R (demand).

2. Each class/flow will have a *state* associated with it, based on the class and flow characteristics such as the AR, MR, CR and R parameters, specified above. Each class/flow can be in one of the three states:

   (a) *congestion state* if $AR<R<=CR$.

(b) *stable state* if $MR<R<=AR$ and

(c) *minimal threshold state* if $R=MR$.

For the adapter of the congested overlay router to control the transient congestion, it needs to estimate the congestion locally, perform BW reassignment and coordinate with adapters (section 5.2) of other nodes.

Local Congestion Estimation and BW Reassignment:

Overlay Router/PGW estimates congestion locally from the state of each flow sharing its outgoing BW. When all the flows at an overlay router/PGW are in *stable state* or *min threshold state*, the router/PGW node is in *stable state* without any congestion. When any of the flows is in *congestion state*, the router/PGW node's adapter does congestion control through BW reassignment.

Our *BW reassignment* algorithm, shown in **Algorithm 2**, considers all flows which are in congested state. The BW reassignment algorithm selects a list of victim LRT & BE flows for each HRT flow i (i.e., LRT & BE flows whose BW allocation is reduced in order to increase BW and protect real-time performance of the HRT flow i). The list of victim flows for HRT flow i is selected based on the *priority* P of flows and the *correlation* among the flows and kept in the data structure *M(i)*. The correlated flows information is obtained from the controller unit. For example, from Figure 5.3, *M(f1)* contains $f3$, $k3$, $f2$ i.e., when $f1$ gets congested, the flows in *M(f1)* transfer their BW share to $f1$. BE flows $k3$ and $f3$ are preferred as victim BE flows for HRT $f1$ over LRT $f2$. $f3$ is preferred over $k3$ as $f1$ and $f3$ are correlated. For each HRT congested flow i (in the order of class priority), the BW shares of LRT & BE victim flows in M(i) (ensuring Min Threshold rate), are transferred to satisfy HRT flow i's need.

For example, let the assured rate (AR) for the flows be $f1$ - 20Mbps, $f2$ - 10Mbps, $f3$ - 10Mbps, $k1$ - 20Mbps and $k3$ - 5Mbps. Let the Min threshold rate for the flows be $f1$ - 10Mbps, $f2$ - 5Mbps, $f3$ - 2Mbps, $k1$ - 10Mbps and $k3$ - 2Mbps. Now, if $f1$ increases its rate to a higher rate 25Mbps, the bandwidth needs to be reassigned to satisfy the increased demand of HRT flow $f1$. *needed* bandwidth is 5Mbps. *M(f1)* contains the ordered victim flow lost for flow $f1$. To satisfy the increased demand, we first look at the victim flow $f3$. $f3$ currently has a share of 10Mbps and its min threshold is 2Mbps. Therefore, we transfer 5Mbps share from $f3$ to $f1$. The new rates now become $f1$ - 25Mbps, $f2$ - 10Mbps, $f3$ - 5Mbps, $k1$ - 20Mbps and $k3$ - 5Mbps.

Algorithm 2 Bandwidth Reassignment Algorithm

```
for each HRT flow i in congested state (priority order) do
  for each victim flow b in M(i) do
    needed ⇐ i.R − i.AR
    oldb = b.AR
    b.AR ⇐ max(b.MR, b.AR − needed)
    needed− = (oldb − b.AR)
    i.AR+ = (oldb − b.AR)
    if needed ≤ 0 then
      break
    end if
  end for
end for
```

Congestion Notification Protocol (section 5.2) informs neighboring adapters about the changed BW shares of all flows.

5.2 Congestion Notification Protocol

We consider an ATM-like back pressure[18] rate control algorithm in our CCC framework. When the publisher (PMU) increases its sending rate pertaining

to a critical event, it introduces more traffic into the network than reserved for its flow. Depending on the availability of excess BW in the downstream PGWs/overlay routers, some nodes can experience congestion because of the increased PMU flow rate. The congested nodes control the congestion by BW reassignment. With BW reassignment the high priority flow gets required BW share, while the low priority victim flows get congested. By this we protect the real-time guarantees of the PMU flows. Explicit congestion notification to the low priority victim flows' senders takes the system back into stable state. This notification is done through the cooperative protocol between each nodes' adapter. Whenever, at a node congestion is estimated and BW is reassigned, the adapter of the node sends a control packet to its upstream node with the flows' new weights. The adapter of the upstream node triggers its BW reallocation according to the new weights in the control packet and notifies its upstream adapter about the flows. This proceeds on till the sender (BE App) which responds to the control packet by decreasing/increasing its sending rate. We use TCP for the reliable delivery of control packets and UDP for the data packets (as we do not need retransmissions, acknowledgements etc.).

Let us explain the protocol with the help of an example in Figure 5.4. When a critical event occurs in Utility A, the A-HRT PMU increases its sending rate. This leads to congestion at the overlay router that is shared between Utility A and Utility B flows. The overlay router through its BW reassignment algorithm when identifies congestion, selects victim flows and recalculates the link BW sharing weights. Based on the spatial correlation policy A-BE is selected as the victim flow. Adapter of the overlay router notifies the new weights to its cooperative upstream adapter (PGW-A). PGW-A responds to the notification by reassigning its outgoing BW and communicat-

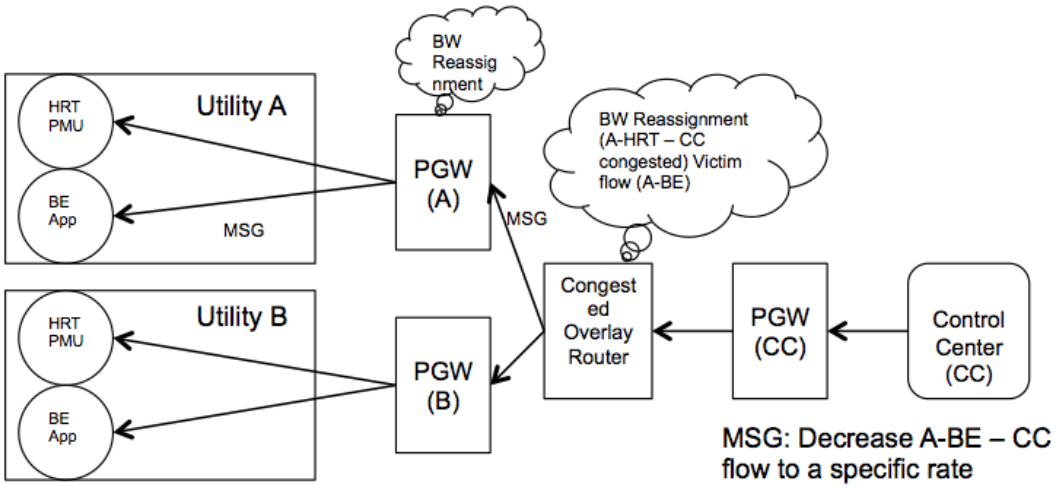

Figure 5.4: Congestion Notification Protocol

ing with the cooperative upstream sender A-BE App with the control packet. The sender A-BE App reacts by decreasing its sending rate. Similarly when a subscriber (CC) wants to receive A-HRT PMU - CC flow at increased rate it notifies to its PGW that follows the above steps, with reassignment done along the path and the senders (PMU and App) notified about their new sending rates.

CHAPTER 6

IMPLEMENTATION

6.1 Overview

We have implemented the overlay router components and the notification protocol on linux machines with object-oriented C++ programming language. The requirement is to have a kernel version >= *linux* 2.6.25 to ensure all kernel modules exist. *iproute2* package's *tc* modules are used to interact with the kernel. These modules help in setting up the kernel network queueing parameters and classifier filters. We use a hierarchical token[22] bucket approach rate limit each flow's bandwidth usage to its reserved rate. Kernel support is needed for packet real time scheduling and BW reassignment. An image of the Kernel multiple service queues is created which is used by the user module to periodically query the statistics, used for the Bandwidth reassignment.

The Phasor Gateway/Overlay Router implementation diagram is shown in Figure 6.1 Now let us discuss the implementation of each component mentioned earlier.

6.2 Controller

This is the main unit that acts as a parent thread and creates all the other unit threads. The main responsibilities include:

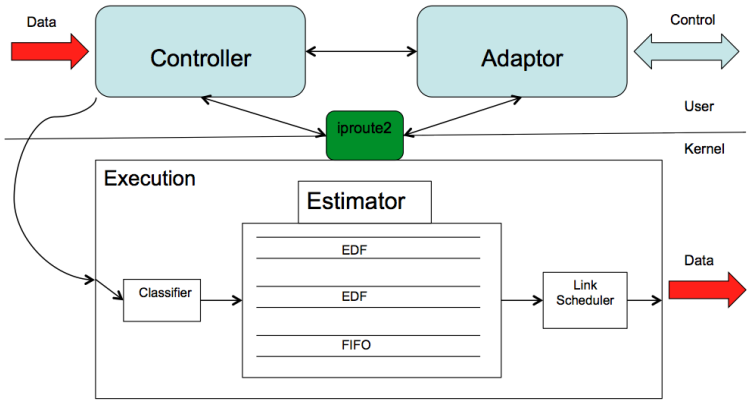

Figure 6.1: Phasor Gateway Implementation

1. Setting up connections with other nodes and broker.

2. Creating the Kernel queues and filters.

3. Reserving the bandwidth and setting up the overlay routing tables.

4. Creating other units.

5. Storing the metadata of the flows.

6. Alarm Messages to broker.

7. Starting the data connection threads one for each connection.

During the system initialization, Controller units are created on each machine. Control connections are setup between QoS broker and the controller module of each Phasor Gateway and Overlay router. When the flow set up decision is done by the brokers of the management plane, brokers inform all the controllers of the gateways/routers in the flow path about the reservation, QoS , flow correlation metadata etc. Controller starts other modules on its machine and sets up the reservation tables on per-flow basis. Metadata of

the flows is the information about the spatial correlation about the same substation flows. Controller also maintains information about the Min Threshold rate of each registered flow and the previous hop overlay node of each flow. This information, obtained from the broker, is used by the adapter module for bandwidth reassignment and congestion notification during congestion. When the BW reassignment algorithm fails to control the congestion alarm messages are sent to the broker as local adaptation is no longer possible. For real-time flow scheduling, we need to set up the kernel queues and filters.

6.3 Data Connection Module

This modules handles the data flows. Currently there is one thread that implements this module. This thread is responsible for listening for data on the UDP port, and send the data through class-specific (HRT, LRT, BE) kernel queue. We plan to extend this module with multi-threaded design. We want to have one thread that handles each flow connection. The multi-threaded implementation helps in further extensions of the framework with Dynamic Soft Real Time scheduler (DSRT). With DSRT[9] the CPU scheduling can be prioritized so that the RT Class A flows get the CPU preference over Class B over BE Class C.

6.4 Execution

Execution unit implements the kernel networking modules. Let us first introduce the *iproute2* package and the *qdisc* scheduler.

6.4.1 Overview of Packages

iproute2 & qdisc:

qdisc: A *qdisc* is a scheduler. Every output interface needs a scheduler of some kind, and the default scheduler is a FIFO. Other *qdiscs* available under Linux will rearrange the packets entering the scheduler's queue in accordance with that scheduler's rules. The *qdisc* is the major building block on which all of Linux traffic control is built, and is also called a queuing discipline. There are different *qdiscs* - *root qdisc* for the *egress* and *ingress qdisc*. Each interface contains both. The primary and more common is the *egress qdisc*, known as the *root qdisc*. It can contain any of the queuing disciplines (*qdiscs*) with potential classes and class structures. Traffic transmitted on an interface traverses the *egress* or *root qdisc*.

iproute2: *iproute2* is a suite of command line utilities which manipulate kernel structures for IP networking configuration on a machine. Of the tools in the *iproute2* package, the binary *tc* is the only one used for traffic control.

As *tc* interacts with the kernel to direct the creation, deletion and modification of traffic control structures, the binary needs to be compiled with support for all of the qdiscs you wish to use. In particular, for the classful *qdiscs*. The classful *qdiscs* can contain classes, and provide a handle to which to attach filters.

```
Usage: tc [ OPTIONS ] OBJECT { COMMAND | help }
where  OBJECT := { qdisc | class | filter }
       OPTIONS := { -s[tatistics] | -d[etails] | -r[aw] }
```

As shown above *tc* can be used to create *qdisc, class* and *filter*. It can also be used to obtain the statistics of the queuing module.

6.4.2 Classifiers

Classifiers sort or separate traffic into queues. Classifying is the mechanism by which packets are separated for different treatment, possibly different output queues. During the process of accepting, routing and transmitting a packet, a networking device can classify the packet a number of different ways. In our system, packets contain the class information such as High Real Time(HRT), Low Real Time(LRT) and Best Effort(BE). Based on this information classifiers puts the packet into the appropriate class queue of the *root qdisc*. Classification can also be done based on Source port, IP, Destination port, IP.

Filters are one of the key elements of traffic control. Filters execute the role of classifiers. Linux filters allow the user to classify packets into an output queue with either several different filters or a single filter. Filters can be attached either to *classful qdiscs* or to classes, however the enqueued packet always enters the *root qdisc* first. After the filter attached to the root qdisc has been traversed, the packet may be directed to any subclasses (which can have their own filters) where the packet may undergo further classification. Filter objects, which can be manipulated using tc, can use several different classifying mechanisms, the most common of which is the *u32* classifier. The *u32* classifier allows the user to select packets based on attributes of the packet.

6.4.3 Queuing Strategies

Our main goal is to do priority scheduling with certain kind of bandwidth guarantees for all classes. We use a Hierarchical Token Bucket (HTB) filter kind of approach. HTB[22] uses the concepts of tokens and buckets along

with the class-based system and filters to allow for complex and granular control over traffic. With a complex borrowing model, HTB can perform a variety of sophisticated traffic control techniques. One of the easiest ways to use HTB immediately is that of shaping. This queuing discipline allows the user to define the characteristics of the tokens and bucket used and allows the user to nest these buckets in an arbitrary fashion. When coupled with a classifying scheme, traffic can be controlled in a very granular fashion.

Now we would explain the creation of the HTB *root qdisc*, and associating classes with it.

```
Usage: tc qdisc add dev eth0 root htb [default N] [r2q N]
```

```
Usage: tc  class add ... htb rate R1 burst B1 [prio P]
    [slot S] [pslot P] [ceil R2] [cburst B2]
    [mtu MTU] [quantum Q]
```

Let us explain each of the parameters passed to the above commands.

default: An optional parameter with every HTB *qdisc* object, the default default is 0, which cause any unclassified traffic to be dequeued at hardware speed, completely bypassing any of the classes attached to the *root qdisc*.

rate: Used to set the minimum desired speed to which to limit transmitted traffic. This can be considered the equivalent of a committed information rate (CIR), or the guaranteed bandwidth for a given leaf class.

ceil: Used to set the maximum desired speed to which to limit the transmitted traffic. The borrowing model should illustrate how this parameter is used. This can be considered the equivalent of "burstable bandwidth".

burst: This is the size of the rate bucket (see Tokens and buckets)[22]. HTB will dequeue burst bytes before awaiting the arrival of more tokens.

cburst: This is the size of the ceil bucket (see Tokens and buckets)[22]. HTB will dequeue *cburst* bytes before awaiting the arrival of more *ctokens*.

quantum: This is a key parameter used by HTB to control borrowing. Normally, the correct quantum is calculated by HTB, not specified by the user. Tweaking this parameter can have tremendous effects on borrowing and shaping under contention, because it is used both to split traffic between children classes over rate (but below ceil) and to transmit packets from these same classes.

r2q: Also, usually calculated for the user, *r2q* is a hint to HTB to help determine the optimal quantum for a particular class.

mtu: Typical size of the packet that the underlying network can send.

prio: Priority of the Class.

Now we explain the actions associated with each command that creates *qdisc, class* and *filter*.

```
tc qdisc add (1) dev eth0 (2) root (3) handle 1:0 (4) htb (5)
```

1. Add a queuing discipline. The verb could also be delete.

2. Specify the device onto which we are attaching the new queuing discipline.

3. This means "egress" to *tc*. The word root must be used, however. Another *qdisc* with limited functionality, the *ingress qdisc* can be attached to the same device.

4. The handle is a user-specified number of the form major:minor. The minor number for any queueing discipline handle must always be zero (0). An acceptable shorthand for a *qdisc* handle is the syntax "1:", where the minor number is assumed to be zero (0) if not specified.

5. This is the queuing discipline to attach, HTB in this example. Queuing discipline specific parameters will follow this. In the example here, we add no qdisc-specific parameters.

Above was the simplest use of the *tc* utility for adding a queuing discipline to a device. Here's an example of the use of *tc* to add a class to an existing parent class.

```
tc class add (1) dev eth0 (2)  parent 1:1 (3) classid 1:6(4)  htb(5)
rate 256kbit (6) ceil 512kbit (7)
```

1. Add a class. The verb could also be delete.
2. Specify the device onto which we are attaching the new class.
3. Specify the parent handle to which we are attaching the new class.
4. This is a unique handle *(major:minor)* identifying this class. The minor number must be any non-zero (0) number.
5. Both of the classful *qdiscs* require that any children classes be classes of the same type as the parent. Thus an HTB qdisc will contain HTB classes.
6. Class specific reserved rate.
7. Class specific maximum rate.

```
tc filter add (1) dev eth0 (2)  parent 1:0 (3) protocol ip (4)
prio 5 (5) u32 (6) match ip port 22 0xffff  (7)
match ip tos 0x10 0xff  (7) flowid 1:6 (8)
```

1. Add a filter. The verb could also be delete.

2. Specify the device onto which we are attaching the new filter.

3. Specify the parent handle to which we are attaching the new filter.

4. Specify to match only the IP packets.

5. The prio parameter allows a given filter to be preferred above another.

6. This is a classifier, and is a required phrase in every *tc* filter command.

7. These are parameters to the classifier. In this case, packets with a type of service flag (indicating interactive usage) and matching port 22 will be selected by this statement.

8. The *flowid* specifies the handle of the target class (or *qdisc*) to which a matching filter should send its selected packets.

Control Queue

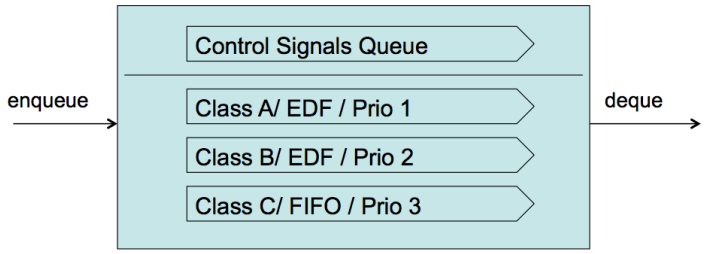

Figure 6.2: Control and Data Packet Queuing

We certainly want control packets to be prioritized over any of the data packets. We use HTB class-based queueing for the data packets. If there is a control packet waiting for the service, it should be the first packet to be served. Hence we use a priority queueing between the *Control queue* and

the *HTB root qdisc*. In the Figure 6.2 the enqueue and dequeue take place following the priorities and the rules of the class-based queueing used.

Queue Buffer

The size of the queue can be controlled by the *tc* commands. Certainly there is a tradeoff between end-to-end latency and the buffer overflow drop rate. If we maintain large queues than that can be serviced, the packets at the end of the queue experience large delays resulting in missing real time latency deadlines. Hence, we adapt the queue size to the rate of the class using the queue. We know that each class is bounded in its bandwidth usage. Hence, if we maintain a large queue, but with Random Early Detection (RED) it certainly at the midpoint of the trade-off between drop rate and latency. The packets that need to be at the head of the queue are Earliest Deadline First Packets for the real-time queues and First-in Packets for the best effort queues. Hence, as shown in Figure 6.2 the Class A and Class B queues can use EDF routines. We hook these routines from the iEDF module of the iDSRT.

6.4.4 Link Scheduler

Figure 6.3 shows the flow of dequeue on the data packets at any Overlay Router/Phasor Gateway. When the network interface hardware is ready to send a packet *dequeue* routine is called on the *root qdisc*. Based on the Class hierarchy and the reserved rate parameters, one of the classes is called with the dequeue routine. The class in turn calls the dequeue routine on the leaf queue which holds the actual packets. As mentioned previously the Real Time queues' enqueue/dequeue routines can be replaced with the

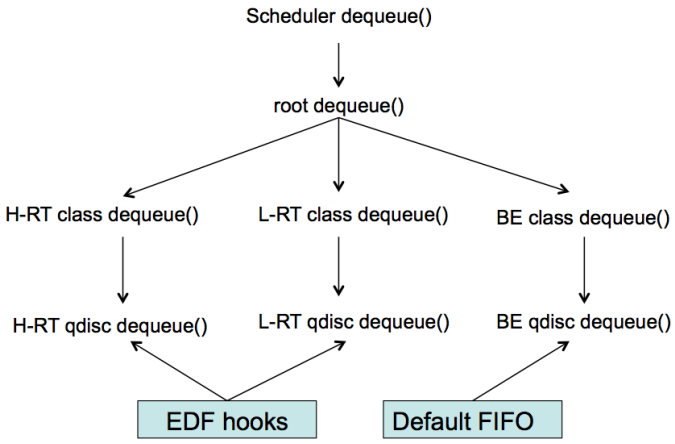

Figure 6.3: Link Scheduler

EDF routines and the Best Effort queue executes the default first-in-first-out routines.

6.4.5 Estimator

We implement a conservative estimator. As, we are dealing with transient congestion and our goal is to protect the real-time guarantees of PMU flows during this small period, the estimator needs to consider the recent events over the history. Therefore, our weight is chosen in such a way that it gives more weightage to the instantaneous component and less weight to the history component while estimating the sent rate and backlogged data of each flow.

6.5 Adapter

For the adapter of the congested overlay router to control the transient congestion, it needs to estimate the congestion locally, perform BW reassignment and coordinate with adapters of neighboring nodes.

6.5.1 Local Congestion Estimation

Congestion is estimated locally from the state of each flow sharing its outgoing BW. The state of the flow is dependent on the flow characteristics like AR, MR, CR and R. This information about the reserved rate, minimal rate, ceil rate and the incoming rate for each flow can be obtained from the kernel network queues. We use the *tc* interface to obtain the statistics, parse it and store it in the above data structure (image of the kernel network queue), which is used for the congestion estimation.

```
tc -s show class dev eth0
```

The above command shows statistics of the entire output interface queues. Our perl scripts extract the needed information and store each class' information into its structure as shown below.

Class Hierarchy Image

```
struct stats_class{
   uint32_t class_id;
   uint32_t flow_id;
   long sent_bytes;
   long last_sent_bytes;
   struct timeval tv_last;
   struct timeval tv_now;
   uint32_t assured_rate;
   uint32_t ceiling_rate;
   uint32_t dropped_packets;
   uint32_t last_dropped_packets;
   uint32_t usage_rate;
```

```
    uint32_t last_usage_rate;
    uint32_t usage_pps_rate;
    uint32_t last_usage_pps_rate;
    uint32_t last_backlog_bytes;
    uint32_t backlog_bytes;
    uint32_t last_backlog_packets;
    uint32_t backlog_packets;
};
```

1. *class_id*: The identifier of the class used to identify HRT, LRT and BE classes.
2. *flow_id*: A unique global identifier to identify the flow in the system.
3. *sent_bytes&last_sent_bytes*: Calculates the amount of data sent over the last period.
4. *assured_rate*: Current Reserved Bandwidth.
5. *dropped_packets&last_dropped_packets*: Calculates the number of dropped packets of this class/flow over the last period.
6. *backlog_packets&last_backlog_packets*: Calculates the number of backlogged packets of this class/flow over the last period.
7. *backlog_bytes&last_backlog_bytes*: Calculates the number of backlogged bytes of this class/flow over the last period.
8. *usage_rate*: BW usage rate in bytes per sec (Estimated using EWMA).
9. *usage_pps_rate*: BW usage rate in packets per sec (Estimated using EWMA).

Congestion over the last period of time is estimated using the number of dropped packets, backlogged packets, sent bytes , assured rate etc. from the above data structure over the last period of time. Time period is taken to be *1 sec.* i.e., For every 1 sec, the above data structure is filled with new data/statistics about the state of the flows, queues in the kernel and then the congestion estimation takes place.

Once there is congestion estimated, in at least one of the HRT flows, Bandwidth reassignment is to be done.

6.5.2 Bandwidth Reassignment

Bandwidth Reassignment Algorithm needs the following inputs

1. Correlation of flows metadata data structure from the controller module.

2. The Class Hierarchy image data structure with the statistics/state of the kernel network queues, flows.

3. Data structure that stores the minimal threshold rate for each class/flow.

BW Reassignment Algorithm executes the following steps.

1. All congested HRT flows are considered for treatment in the class priority order, flows that belong to same class are considered in arbitrary order.

2. For each congested HRT i, it first selects the victim flows. Victim flow selection is dependent on the priorities of flows and the correlation among the flows. i.e., The selection starts with the lowest priority flows, and from flows with the same priority, the one that is most correlated with the congested HRT flow is selected as the preferred victim.

3. For each victim flow selected above (corresponding to congested HRT flow i), it decreases the BW share of the victim flow, making sure that the min threshold rate is obeyed. This share is transferred to the congested HRT flow i.

4. The above step repeats for all the victim flows corresponding to the congested HRT flow i, till the need of HRT flow i is satisfied.

5. Repeats steps 2-4 for all the congested HRT flows.

Once the flows have their new weights the *tc qdisc/class* interface is used to change the parameters of the kernel network queues. These new weights of flows are to be coordinated to the neighboring nodes which is done by the congestion notification protocol.

6.6 Congestion Notification Protocol

Congestion Notification Protocol will require the data structure that provides information about the previous hop Overlay Router/PGW corresponding to each flow. This structure is available with the controller module as it the one responsible for initial flow set-up phase. Once the BW reassignment algorithm transfers the BW allocation of low-priority victim flows to the congested HRT flows, the notification protocol will extend the congestion information to the neighboring nodes and they can cooperative coordinate among themselves to control congestion and protect the real-time guarantees of HRT PMU flows.

We have seen that a control queue is maintained, which is different from the class-based data queues. This control queue is given the highest priority so as to make sure that the control packets are sent as quickly as possible.

TCP is used for sending the control packets as it needs to be reliable.

The receiving overlay router/PGW of the control packet, even though congestion is not estimated locally, obeys the control packet, executes the bandwidth reassignment algorithm and runs the notification protocol to notify its neighboring nodes.

CHAPTER 7

EVALUATION

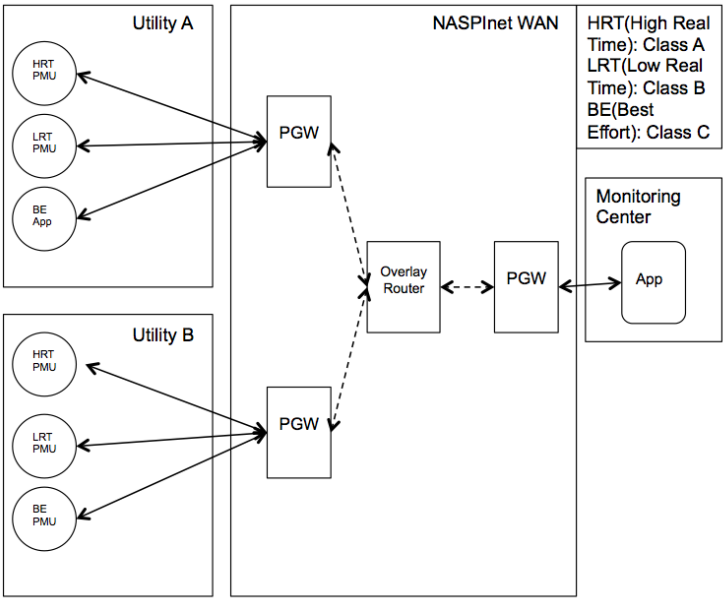

Figure 7.1: Test Scenario

Our testbed constitutes 2 substations with 3 sensors each(1 HRT PMU, 1 LRT PMU, 1 BE App), three phasor gateways, one overlay router and 1 control center as shown in Figure 7.1.

7.1 Scenario

We evaluate the Cooperative congestion control framework in the above test scenario where utility A and utility B with 3 sensors each are connected to the Monitoring center via the NASPInet WAN. Congestion can occur at

Phasor Gateways(PGWs) and Overlay Routers.

7.2 Parameters

For simplicity all the sensors generate sample packet of average size 260 bytes (phasor fields and digital channel fields)[1]. Service class queues are maintained at PGWs/overlay routers to hold 20 data packets of each flow. The sensors can operate at a frequency ranging from 1Hz to 60Hz. The minimum threshold bandwidth for the classes are HRT - 5200 Bytes, LRT - 2600 Bytes , BE - 600 Bytes. End-to-end deadlines assumed for Class A HRT flows is 50ms and Class B LRT flows is 100 ms. The broker response/global adaptation time is taken to be 60 sec (the goal is to protect HRT flows during this period when the system is not in stable state).

7.3 Test

At *t=0* all flows start with an initial frequency of 20Hz. At *t=40* sec the Utility A's HRT PMU sensor increases its frequency to 40Hz. Later at *t=60* sec Utility B's HRT PMU sensor increases its frequency to 40 Hz. At *t=80* sec Utility A's HRT PMU and Utility B's HRT PMU increase their frequencies to 50 Hz. Finally, at *t = 90* sec Utility A's HRT PMU and B-HRT PMU increase their frequencies to 60 Hz.

7.4 Discussion

Figure 7.2 , 7.3, 7.4 show the latency, throughput and drop rate when the cooperative congestion control framework is not used i.e the case just with global adaptation waiting for decision from the broker. Figure 7.5 , 7.6, 7.7

Flow	$0-40$	$40-60$	$60-80$	$80-90$	$90-100$
A-HRT	0%	50%	50%	40%	33%
A-LRT	0%	0%	0%	0%	0%
B-HRT	0%	0%	50%	40%	33%
B-LRT	0%	0%	0%	0%	0%

Table 7.1: Percentage of Deadline Missed Packets without Cooperative congestion control framework

Flow	$0-40$	$40-60$	$60-80$	$80-90$	$90-100$
A-HRT	0%	4%	0%	1.8%	0%
A-LRT	0%	0%	0%	1%	0%
B-HRT	0%	0%	3%	1%	0%
B-LRT	0%	0%	0%	1%	0%

Table 7.2: Percentage of Deadline Missed Packets with Cooperative congestion control framework

show the same with cooperative congestion control in place. Table 7.1 shows the percentage of real-time packets with respect to sent number that missed their deadlines without the framework. Table 7.2 shows the percentage of real-time packets that missed their deadlines with the framework. Table 7.3 shows the percentage of real-time packets that are dropped due to buffer overflow without the framework. Table 7.4 shows the percentage of real-time packets that are dropped due to buffer overflow with the framework. The sent number of packets is the sum of overflow dropped packets, deadline missed packets and legally sent packets.

From *t=0* to *t=40* sec all the flows are in stable state. At *t=40* sec the A-HRT flow doubles its frequency. When our framework is not used, all the extra packets above reserved rate are dropped and the also the deadlines of the sent packets are missed as shown in TABLE 7.1. When the CCC framework is used, A-BE and B-BE are selected as the victim flows and BW reassignment/congestion notification happen which dramatically decreases

Flow	$0-40$	$40-60$	$60-80$	$80-90$	$90-100$
A-HRT	0%	50%	50%	60%	66%
A-LRT	0%	0%	0%	0%	0%
B-HRT	0%	0%	50%	60%	66%
B-LRT	0%	0%	0%	0%	0%

Table 7.3: Percentage of Buffer Overflow dropped packets without Cooperative congestion control framework

Flow	$0-40$	$40-60$	$60-80$	$80-90$	$90-100$
A-HRT	0%	0%	0%	0%	16%
A-LRT	0%	0%	0%	0%	0%
B-HRT	0%	0%	0%	0%	16%
B-LRT	0%	0%	0%	0%	0%

Table 7.4: Percentage of Buffer Overflow dropped packets with Cooperative congestion control framework

the deadline miss rate and the overflow drop rate. Later at *t=60* sec the B-HRT flow doubles its frequency. B-BE is selected as the victim flow and BW reassignment/notification occurs. Both the A-BE and B-BE flows are decreased to their minimum threshold 600 Bytes. At *t=80* sec both A-HRT and B-HRT increase their frequency to 50 Hz. In the case of no CCC framework usage we can see all packets getting dropped due to buffer overflow, whereas with framework A-LRT and B-LRT are selected as victims. Both A-LRT and B-LRT reach to their min threshold 2600 Bytes. At *t=90* sec both A-HRT and B-HRT increase their frequencies to 60 Hz. As per the BW reassignment algorithm there are no more victim flows and hence all the extra packets get dropped. Finally, at *t=100* sec the global adaptation by the broker happens and the system again goes into stable state. Our CCC framework made possible to achieve the real-time guarantees of the PMU flows during the transient instability period (between 40th and 100th sec).

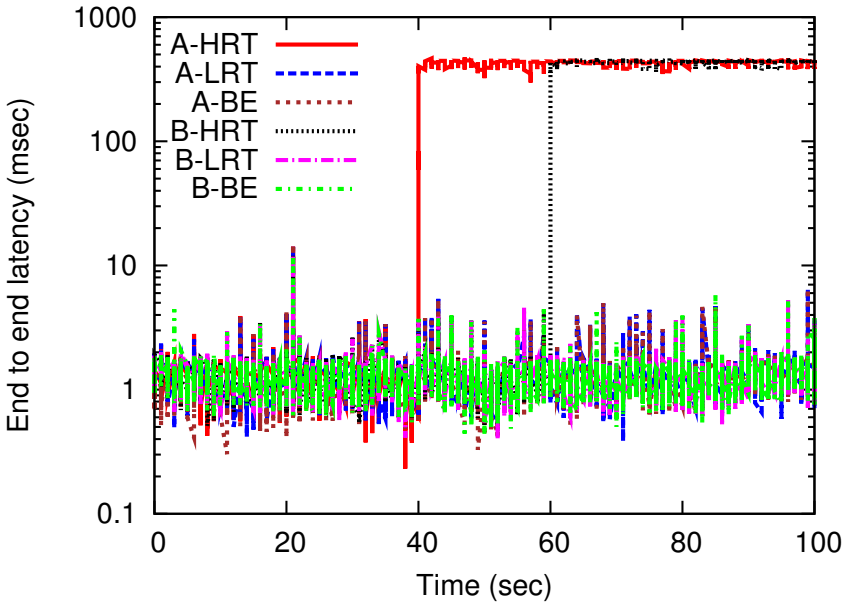

Figure 7.2: End to end latency, without using the framework

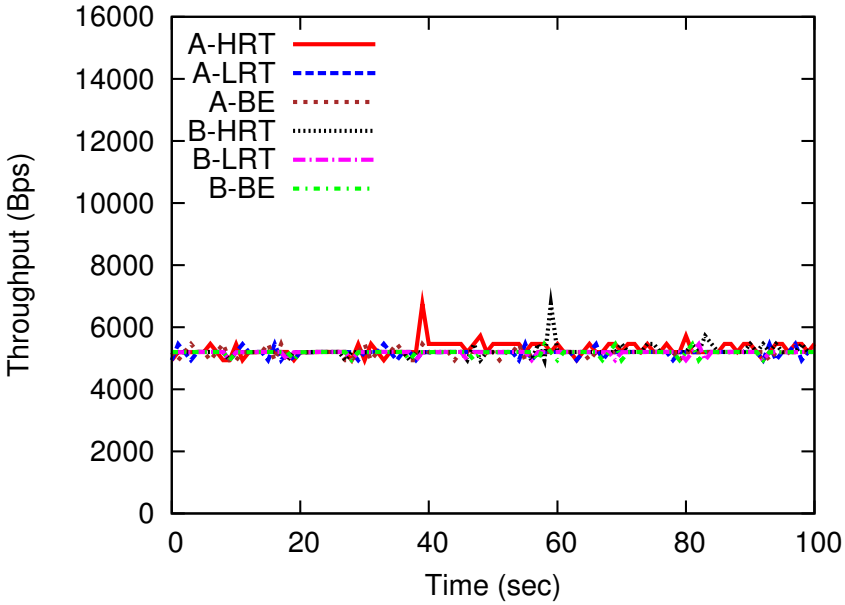

Figure 7.3: Throughput without using the framework

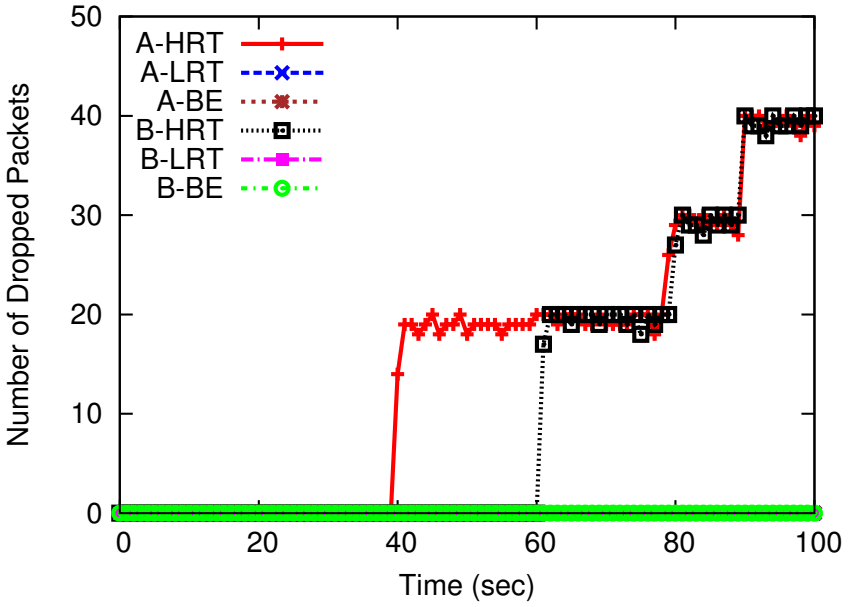

Figure 7.4: Dropped packets due to buffer overflow without using framework

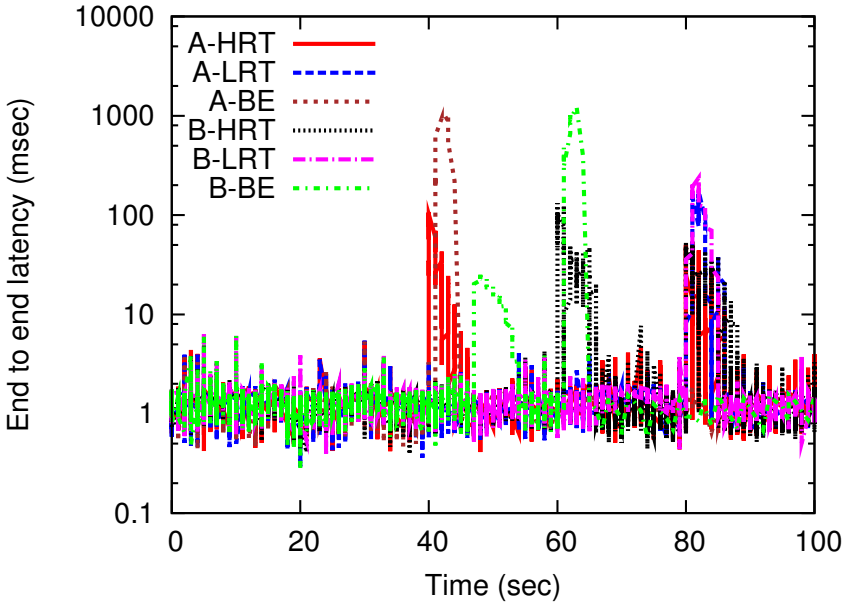

Figure 7.5: End to end latency, with Cooperative congestion control

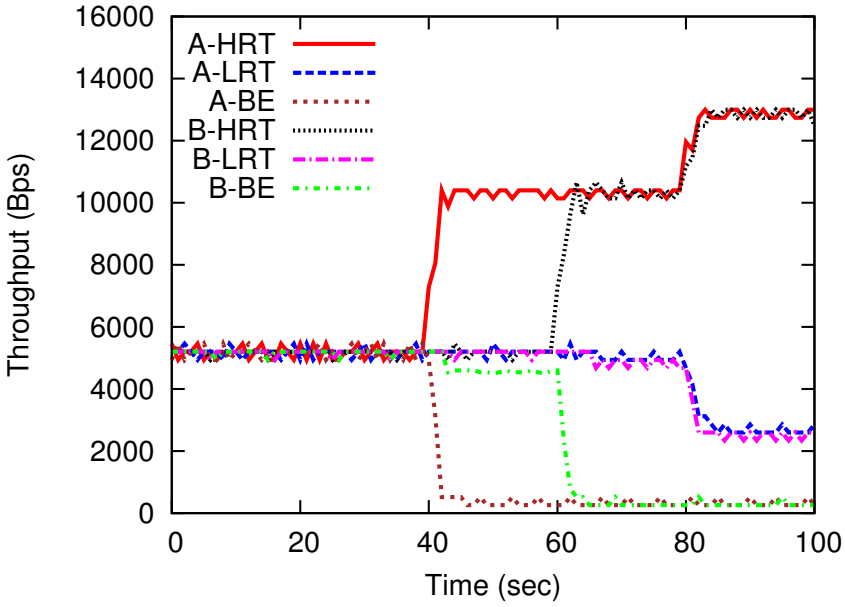

Figure 7.6: Throughput, with Cooperative congestion control

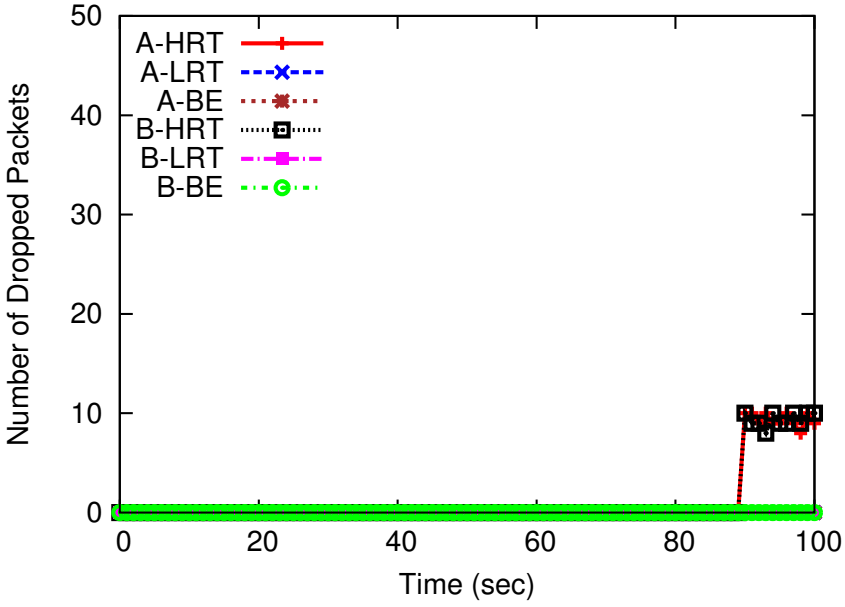

Figure 7.7: Dropped packets due to buffer overflow with Cooperative congestion control

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

In this thesis we presented Cooperative Congestion Control framework to ensure real-time guarantees during transient changes in real-time network traffic from Phasor Measurement Unit (PMU) sensors. The framework utilizes a) NASPI[17] aligned multiple service class queuing architecture; b) Cooperative real-time flow scheduling and Bandwidth reassignment; c) Cooperative coordination and back-pressure among neighboring nodes; to yield real-time PMU data guarantees during transient traffic pattern changes and/or overload situations. We have designed and implemented the two main components that constitute the framework.

1. Overlay router/Phasor Gateway modules that do prioritizing of flows, real time packet scheduling, bandwidth reassignment adaptation.

2. Early congestion notification protocol to notify the congestion and reassigned bandwidth weights to upstream senders.

In the Phasor Gateway design we use multiple queues one for each service class, as explained in the NASPI databus specifications. Real time scheduling and bandwidth reservation for each class is done through the *iproute2* interface, which interacts with the kernel queuing modules. Hierarchical token bucket approach is deployed as the queuing strategy. Our design considered the cooperative nature of nodes and the correlation between the flows, flows from same substation are considered to be correlated. Our control proto-

col runs over TCP and the data connection are maintained over UDP so as to avoid expensive acknowledgements and unnecessary retransmissions. Our experiments confirm the real-time flow performance to be better with the framework in place. Results show that local adaptation (with the CCC framework) along with global adaptation (QoS broker adaptation) maintains stability in the system during the transient periods with no QoS violations.

The lessons learnt while working on this framework are a) Using standardized kernel interfaces to modify the kernel is very important so that compatibility issues with different versions do not occur. b) Changes to Overlay Gateway/ Router Modules are feasible, where as enforcing our modules on physical routers is impractical. c) QoS broker's support is alone not sufficient to ensure real-time guarantees of PMU flows during transient instability and we need local adaptation at the nodes and cooperative control with neighbors. d) Back pressure approaches control congestion very efficiently with little overhead in smaller networks.

As a part of future work we can add the support of DSRT at the CPU level to prioritize threads so that threads that handle real-time flows get the preference. Presently we run our experiments on LAN which is a controlled environment. It would be interesting to see the performance, metrics like latency, drop rate, BW usage when deployed on large scale public networks like planetlab. This framework can be integrated with any Wide Area Network architecture that is designed following NASPI specifications.

REFERENCES

[1] IEEE Standard for Synchrophasors for Power Systems, IEEE C37.118 2005.

[2] www.naspi.org.

[3] North American SynchroPhasor Initiative. (2009, May) Data Bus Technical Specifications for North American Synchro-Phasor Initiative Network. [Online]. Available: http://www.naspi.org/resources/dnmtt/ naspinet/naspinet databus final spec 20090529.pdf

[4] Rakesh Bobba, Erich Heine, Himanshu Khurana and Tim Yardley. Exploring a Tiered Architecture for NASPInet, 1st IEEE PES Conference on Innovative Smart Grid Technologies (ISGT '10), Gaithersburg, Maryland, January 2010.

[5] North American SynchroPhasor Initiative . (2009, May) Phasor Gateway Technical Specifications for North American Synchro-Phasor Initiative Network. [Online]. Available: http://www.naspi.org/resources/ dnmtt/naspinet/naspinet phasor gateway final spec 20090529.pdf

[6] R. E. Schantz, J. P. Loyall, C. Rodrigues, D. C. Schmidt, Y. Kr- ishnamurthy, and I. Pyarali, Flexible and adaptive qos control for distributed real-time and embedded middleware, in Middleware 03: Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware. New York, NY, USA: Springer-Verlag New York, Inc., 2003, pp. 374393.

[7] Bakken, D. E., Hauser, C. H., Gjermundrd, H., and Bose, A. Towards more flexible and robust data delivery for monitoring and control of the electric power grid, 2007.

[8] Bouassida, M.-S., and Shawky, M. A cooperative congestion control approach within vanets: Formal verification and performance evaluation. *Eurasip Journal on Wireless Communications and Networking* – (2010), –.

[9] Hoang Nguyen, Raoul Rivas, Klara Nahrstedt: iDSRT: Integrated Dynamic Soft Real-Time Architecture for Critical Infrastructure Data Delivery over WLAN. QSHINE 2009: 185-202

[10] Floyd, S. Tcp and explicit congestion notification. *ACM Computer Communication Review 24* (1994), 10–23.

[11] Floyd, S., and Jacobson, V. Random early detection gateways for congestion avoidance, 1993.

[12] Vijay Sivaraman and Fabio M. Chiussi and Mario Gerla Deterministic end-to-end delay guarantees with rate controlled EDF scheduling, Performance Evaluation, 2006, Volume 63 pages 4–5.

[13] Vijay Sivaraman and Fabio M. Chiussi and Mario Gerla Traffic Shaping for End-to-End Delay Guarantees with EDF Scheduling, Proc. of IWQoS 2000, pages 10–18.

[14] Floyd, S., and Jacobson, V. Link-sharing and resource management models for packet networks, 1995.

[15] Harrison, D., and Kalyanaraman, S. An edge-to-edge overlay congestion control architecture for the internet, 2001.

[16] Hauser, C. H., Bakken, D. E., Dionysiou, I., Gjermundrd, K. H., Irava, V. S., and Bose, A. Security, trust and qos in next-generation control and communication for large power systems. *International Journal of Critical Infrastructures 2007* (2007).

[17] Hu, Y., Donnelly, M., Helmer, T., Tram, H., Martin, K., Govindarasu, M., Uluski, R., and Cioni, M. Naspinet specification - an important step toward its implementation. In *HICSS* (2010), IEEE Computer Society, pp. 1–9.

[18] Ohsaki, H., Murata, M., Suzuki, Y., Ikeda, Y., and Miyahara, H. Rate-based congestion control for atm networks. *ACM SIGCOMM Computer Communication Review 25* (1995), 60–72.

[19] Ramakrishnan, K. K., and Jain, R. A binary feedback scheme for congestion avoidance in computer networks. *ACM TRANSACTIONS ON COMPUTER SYSTEMS 8* (1990), 158–181.

[20] Risso, F., and Gevros, P. Operational and performance issues of a cbq router, 1999.

[21] Shenker, S., Clark, D. D., and Zhang, L. A scheduling service model and a scheduling architecture for an integrated services packet network. Tech. rep., 1993.

[22] HTB for Linux, http://luxik.cdi.cz/ devik/qos/htb

[23] Stoica, I., Zhang, H., and Ng, T. S. E. A hierarchical fair service curve algorithm for link-sharing, real-time and priority services. ACM, pp. 249–262.