HET 556
Design & Development Project 2

TCAPS
Traffic Classification And Prioritisation System

Project Final Report

SWINBURNE UNIVERSITY OF TECHNOLOGY

CENTRE FOR ADVANCED INTERNET ARCHITECTURES

Student: Lawrence Stewart ID: 2163896

TCAPS Project Final Report

Signatures

Student's Name: Lawrence Stewart

Student's Signature: ____ Date: 25/11/2005

Supervisor's Name: Grenville Armitage

Supervisor's Signature: ____________________________________ Date: 25/11/2005

TCAPS Project Final Report

Table of Contents

TCAPS Project Final Report

1. Executive Summary

The Traffic Classification and Prioritisation System (TCAPS) project aimed to develop an architecture for realtime network traffic classification and prioritisation for use in ISP broadband access networks, as well as a prototype of the system. The system aims to address the issue of traffic Quality of Service (QoS) for realtime/interactive traffic at the network edge i.e. customer to ISP links, which are typically the primary bottleneck in end to end network communication. The system specifically targets the upstream CPE to ISP link, which typically tends to be at least 4 times slower than the downstream speed, making it the primary bottleneck within the CPE/ISP link. The system provides QoS automatically to CPE/ISP network traffic that requires it, doing so without the user or networked applications requiring any knowledge of the underlying system. This allows end users to use the networked realtime/interactive applications they need, without the concern of these applications' performance degrading as other users on the local LAN begin to compete for resources by using other non realtime/interactive networked applications.

The prototype implementation of the system consists of 6 software modules: TCAPS signalling protocol, TCAPS QoS subsystem translator, TCAPS client interface, TCAPS manager, TCAPS flow sifter and TCAPS flow classifier.

The signalling protocol and QoS subsystem translator modules were implemented as procedural C software libraries, in the hope that they may be useful to external individuals/companies wishing to integrate TCAPS code into their software and devices.

The TCAPS client interface, TCAPS manager, TCAPS flow sifter and TCAPS flow classifier were implemented as C++ applications to decrease development time by use of existing language libraries and functionality.

Supporting the software developed as part of the TCAPS project are a number of existing open source software solutions. These solutions provided required functionality without the need to build it from scratch during the TCAPS prototype development.

Unfortunately due to time constraints, insufficient testing of the TCAPS prototype was able to be performed. The prototype was verified for functional correctness and put through some simple stress tests, but these did not conclusively evaluate the operational capabilities of the prototype. Of the minimal testing that was performed, the prototype was found to function according to specification. The flow sifter and flow classifier modules were stress tested with 100Mbit line rate traffic and 10 concurrent flows or less. Packet sifting and classification were found to be able to handle this load running on commodity PC hardware.

Overall, the TCAPS architecture has proven itself to be a robust, scalable and efficient means for providing automated QoS over consumer broadband links. The prototype implemented for the TCAPS project has proven the feasibility of the architecture and feasibility of overall approach to solving the problem of broadband QoS provision.

In order to move the project towards commercialisation, a number of limitations and items of further work would need to be seen to. The most pertinent being:

- Selecting and implementing a means for TCAPS node IP dissemination.
- Implementing ISP side TCAPS module polling to handle node failures.
- Integrating a security framework into or around TCAPS.

- Getting CPE manufacturers on board and developing their own TCAPS enabled CPE.
- Running an ISP based TCAPS trial.

An opportunity for collaboration with industry has been identified, in the form of Ubicom Inc. and their StreamEngine technology. This technology provides a great deal of the functionality required by a TCAPS enabled CPE, and would only need to be modified to accept TCAPS signalling information in order to integrate into the TCAPS architecture. A partnership of some form with Ubicom Inc. would expedite the time to market for TCAPS enabled CPE. This would be a beneficial outcome if the decision was made to commercialise the project.

Overall, the TCAPS project has met it design goals, has a functioning prototype capable of demonstrating a majority of the features important to the TCAPS architecture and is therefore considered to have been successful.

It is the recommendation of this report that the TCAPS architecture is a feasible framework to commercially pursue for the provisioning of automated QoS over consumer broadband links. The current TCAPS prototype implementation met its design requirements, but falls short of commercial viability, and requires additional development work and testing to bring it up to a commercial level. The estimated time frame for the completion of the critical additional work required for TCAPS commercial viability is 6 to 12 months.

2. Introduction, Overview & Scope

With the increase in and affordability of broadband access roll out in the forms of Asymmetric Digital Subscriber Loop (ADSL), Cable Internet and 802.11x infrastructure, comes the real possibility of useable, realtime services to the home. Examples include multiplayer online networked games, streaming audio/video content and voice over Internet Protocol (VoIP). There is also the next generation of high speed broadband network architectures to consider such as ADSL2 and IEEE 802.16 etc. which are going to support high definition digital television on demand and the like, along with the previous suite of services.

However, a heterogenous network traffic environment such as a home or small business LAN makes demands on the underlying network infrastructure that can cause it to become a bottleneck e.g. someone making a VoIP call whilst someone else uploads a large file. The VoIP traffic, being realtime and interactive, is far more sensitive to queuing delays and network jitter than the TCP file transfer. The net result in this scenario is that the phone conversation degrades significantly, whilst no major difference is observed for the file transfer except perhaps for a decrease in the rate of throughput.

So how can we ensure these two types of network traffic can coexist happily? The solution proposed by our solution involves separating them into two groups: realtime/interactive and the rest. Once separated, we can prioritise the first group and ensure any special requirements for this traffic class are met. The process of having this occur dynamically and without any user/administrator or application intervention is a new idea within the problem space.

The Traffic Classification and Prioritisation System (TCAPS) project aimed to develop an architecture for realtime network traffic classification and prioritisation for use in ISP broadband access networks, as well as a prototype of the system. The system aims to address the issue of traffic Quality of Service (QoS) for realtime/interactive traffic at the network edge i.e. customer to ISP links, which are typically the primary bottleneck in end to end network communication. The system specifically targets the upstream CPE to ISP link, which typically tends to be at least 4 times slower than the downstream speed, making it the primary bottleneck within the CPE/ISP link. The system provides QoS automatically to CPE/ISP network traffic that requires it, and does so without the user or networked applications requiring any knowledge of the underlying system. The system is also be able to work in parallel with legacy CPE equipment that is not TCAPS enabled, ensuring incremental uptake by existing ISPs and broadband access users is possible.

The final TCAPS architecture and design consists of 6 core modules: the TCAPS signalling protocol, QoS subsystem translator, client interface, flow sifter, flow classifier, and manager. These modules have been implemented completely in software for speed of development and flexibility purposes. Together, these 6 modules interoperate to provide the TCAPS prototype system capable of addressing the problem outlined above.

Figure 1 illustrates a typical home broadband connection scenario. The customer premises (CP) side of the link is a 100Mbps switched Ethernet LAN, and has 3 separate services being used on it: VoIP, web browsing and online interactive gaming. The CP network is connected to the ISP and Internet via a 512kbps downstream, 128kbps upstream broadband access Internet connection, which is a typical configuration for current broadband access Internet plans [1].

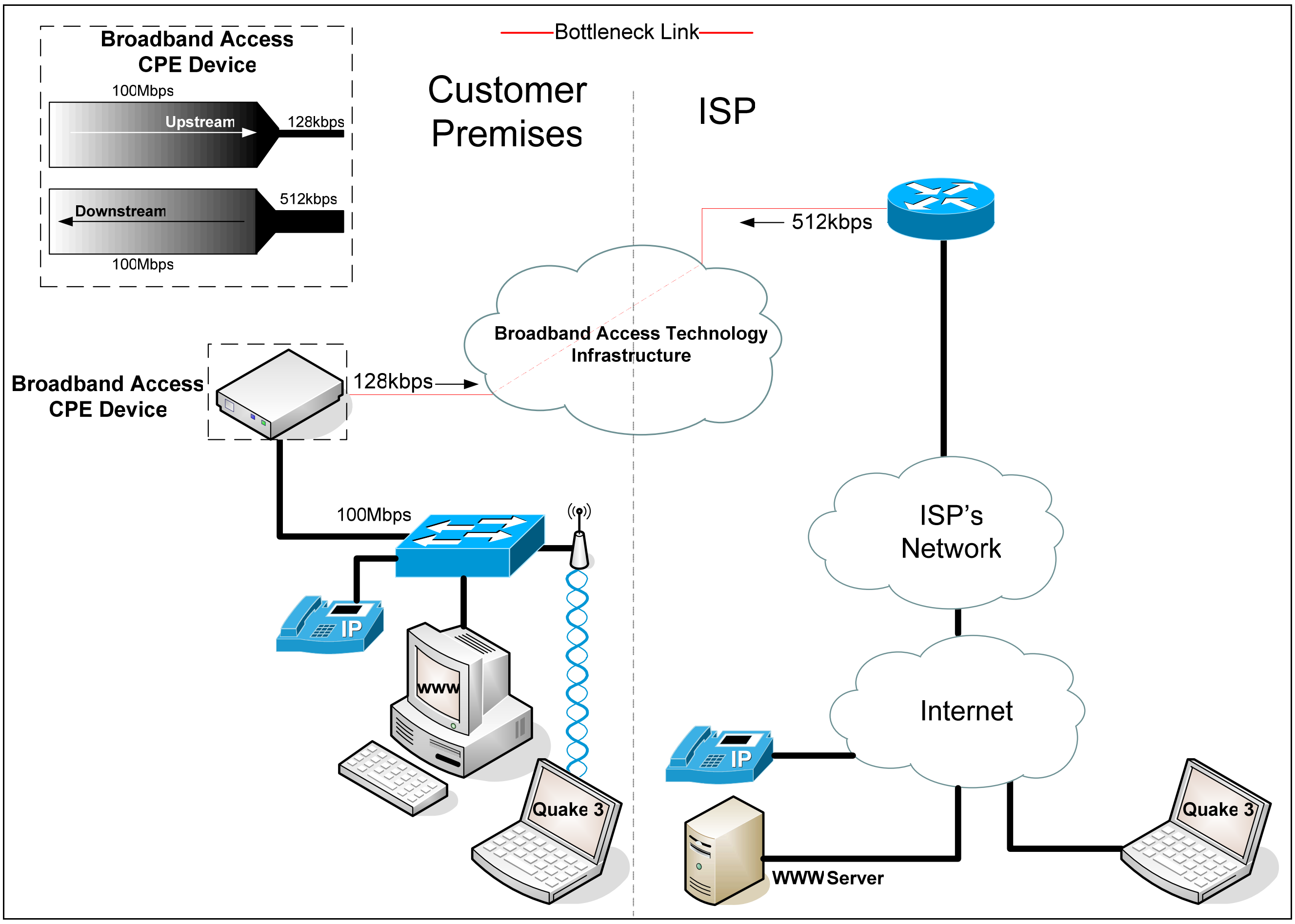

Figure 1. Typical home broadband connection scenario

The major problem being addressed by TCAPS is the bottleneck being caused by the 100Mbps local LAN squeezing traffic onto the 128kbps upstream broadband link, which is 780 times slower than the local LAN. The serialization delay for a 1500 byte packet being sent upstream at 128kbps from the CPE is 93.75ms. This is significant in networking terms, where inter-packet arrival times tend to be sub 100ms. A packet arriving half way through the serialization process will be queued before being sent, and the queue will grow if more packets arrive during the serialization time. This can result in unpredictable packet queuing delays and jitter, caused by waiting in a queue that has heterogenous packets of different sizes in it.

Consider the following example: if a 100 byte packet gets stuck behind a 1500 byte packet that has just started getting serialized onto a 128kbps upstream link, the result is a 94ms wait for the 100 byte packet. Realtime/interactive services such as VoIP and online interactive gaming, tend to rely on small packets (typically well under 500 bytes) being sent at quick, regular intervals [2].

Now consider a realtime/interactive traffic flow sending 100 byte packets upstream at regular intervals of 47ms, with serialization delay ~6ms at 128kbps. A non realtime/interactive flow is also sending 1500 byte packets upstream at random intervals, with serialization delay ~94ms at 128kbps. Figure 2 shows a time sequence diagram of the CPE's upstream traffic queue, with 100Mbps traffic entering from the local LAN, and 128kbps traffic being sent to the ISP/Internet via the broadband access network connection.

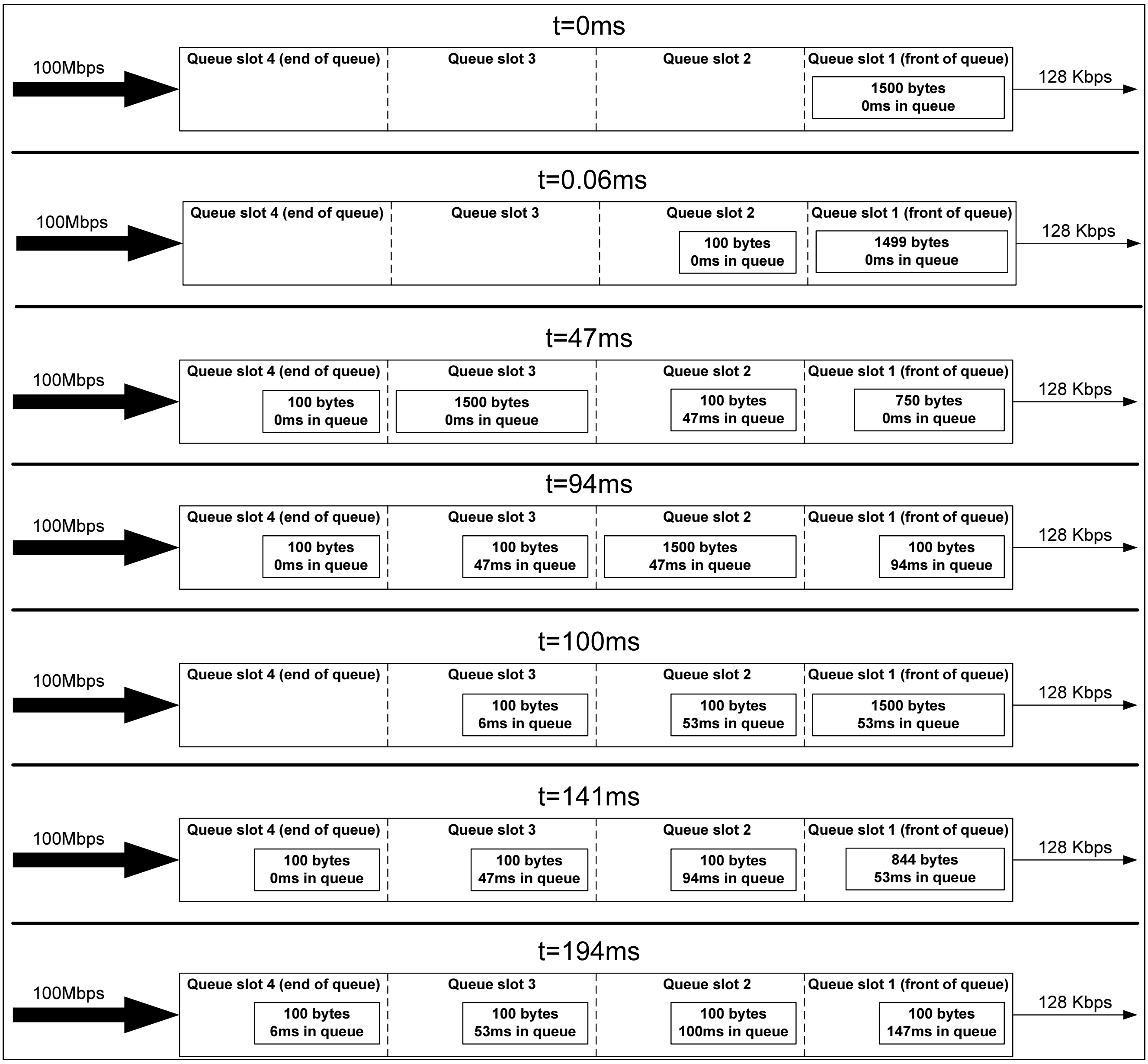

Figure 2. Time sequence diagram for CPE upstream queue

One of the 100 byte packets gets stuck behind a 1500 byte packet that has just started getting serialized onto the upstream link. This leads to a 94ms wait for the 100 byte packet, in which time a 1500 byte packet and two 100 byte packets join the queue. The first 100 byte packet begins being transmitted 94ms after it arrived in the queue, and the 1500 byte packet behind it begins being transmitted 6ms after that. In the 94ms it takes to transmit the second 1500 byte packet, 2 two more 100 byte packets join the queue. The four 100 byte packets are then sent every 6ms until the queue is emptied. The queuing delays for each of the 100 byte packets in order of transmission are 94ms, 147ms, 106ms, 65ms and 24ms.

The end result of this scenario is that a traffic flow that should have consistent 47ms inter-packet arrival times (when not subjected to queuing delays), experiences inter-packet arrival times of 141ms, 194ms, 153ms, 112ms and 71ms. If the realtime traffic flow belonged to a VoIP conversation, for example, these large deviations from the expected inter-arrival times would result in degraded call quality and an unpleasant phone conversation.

Suppose now we established two queues within the CPE for upstream traffic. One of these queues is configured to have delay sensitive realtime/interactive traffic placed in it and the other queue is configured to have all other traffic placed in it. If the delay sensitive queue was configured to have transmission priority onto the upstream link ahead of the other queue, then we would expect realtime/interactive traffic to have reduced queuing times, compared to the example of Figure 2. This premise forms the basis of the TCAPS prototype solution to the problem discussed above, and will be examined in more detail in section 5.2.1 of this report.

Queuing at the ISP for the downstream link is typically not as much of a problem as it as at the CPE for the upstream link. This is because the downstream speeds tend to be much higher (typically at least 4 times greater) than upstream speeds for typical broadband access Internet plans. This means that packets are able to be sent downstream 4 times faster than they can upstream, and the serialization delay is 4 times shorter. The serialization delay for a 1500 byte packet being sent downstream at 512kbps from the ISP is 23.43ms. Even if a packet from a realtime/interactive flow gets queued behind a 1500 byte packet, it will only have to wait 24ms instead of 94ms as before. This observation validates the need to particularly focus on a solution that reduces upstream realtime/interactive network traffic queuing delays and jitter. However, the downstream link would still benefit from a solution that reduced queuing delays and jitter as illustrated in the above example, and is considered as part of the TCAPS architecture.

3. Literature Survey

3.1 IntServ (RFC1633) & RSVP (RFC2205)

Integrated Services (IntServ) [3] and Resource ReSerVation Protocol (RSVP) [4] were defined by the IETF in 1994 and RSVP was revised in 1997 in order to attempt to solve the problem of dynamic QoS provision for realtime/interactive traffic traversing the Internet. The IntServ/RSVP model used signalling protocol messages along the network path between sender and receiver, and each node along the path would store QoS state for the flow requesting guaranteed resources.

The major problem with this approach is that applications wishing to utilise IntServ/RSVP for their QoS requirements need to be written to utilise the RSVP signalling protocol. This would require all realtime/interactive application developers to integrate the RSVP stack into their programs, which is not going to happen. TCAPS is able to perform dynamic QoS provision without the need for the realtime/interactive application to intervene at all, and will work with all existing applications written before the development of TCAPS, unlike the IntServ/RSVP model.

3.2 DiffServ (RFC2475)

Differentiated Services (DiffServ) [5] was defined in 1998 by the IETF to provide an IP QoS solution for use in autonomous systems. DiffServ works by setting bits that correspond to a QoS traffic class in the IP header DS field, which are then inspecting by DiffServ enabled routers along the path and provided with the QoS specified for that particular QoS traffic class.

The major problem with this approach is that every router along the network path needs to be DiffServ enabled and have the same understanding of which DS field bits correspond to which QoS traffic class to ensure proper QoS is given to the correct packet flows. This limits its usefulness to routing within an autonomous system, where all routers are administered by the same administrator(s).

Internet links tend to be massively over provisioned, which means there is very minimal delay/jitter introduced in the network core. The majority of network delay/jitter comes from the CPE/ISP link, which is the area TCAPS is focused on. TCAPS does not change IP packet fields and does not require every device in the end to end path between the CPE and TCAPS server to inspect the packets in order for QoS to be provided.

3.3 MPLS (RFC3031)

The MultiProtocol Label Switching (MPLS) [6] Architecture was defined by the IETF in 2001 as an architecture to simplify the way routers make packet forwarding decisions. Instead of each router along the network path examining the IP packet header and making a forwarding decision based on this information, MPLS only requires the packet header to be analysed once. MPLS then adds a label (short, fixed length value) to the packet, both of which are transmitted to the next hop, where the label is matched in a lookup table, a new label is added and the packet is sent to the next hop. The label is used as the basis for all forwarding decisions, and allows devices that are not capable of doing IP header inspection or not capable of doing the IP header inspection fast enough to perform routing functionality. One of the benefits of MPLS is that a packet's class of service can be partially or completely inferred from the label, which allows for simplified QoS classification and management.

MPLS still suffers from the same drawbacks as DiffServ, in that every node along the network path has to support MPLS in order to provide QoS based on MPLS labels, and each MPLS enabled device needs to know what MPLS labels map to a particular class of service in order to provide consistent

QoS. An administrator also needs to define the initial classification criteria/policies to determine which types of flows should be assigned a particular MPLS label based on their traditional IP header inspection.

TCAPS's ability to automatically classify realtime/interactive flows and signal the required devices to instantiate appropriate QoS for these flows ensures it is not affected by these problems and is a superior solution for the problem being addressed by the TCAPS project.

3.4 Ubicom Inc. & D-Link Systems Inc.

Ubicom Inc. [7] began marketing a technology titled "StreamEngine" [8] in late 2004, which appears to provide the same functionality as TCAPS on the CPE to ISP portion of the network link. D-Link Systems Inc. have released what they claim to be the first series of "gaming" routers, using what they call "GameFuel" technology [9], which is based on Ubicom's StreamEngine technology. These routers are, as their name suggests, being pushed at the lucrative online multiplayer game market, although the technology is technically capable of providing benefits to other realtime/interactive traffic applications/services as well.

The major difference between TCAPS and StreamEngine is that StreamEngine relies on packet inspection techniques to perform its classification. Packet inspection involves examining the entire contents of an IP packet and inferring what type of traffic is being examined based on information such as well known TCP/UDP port numbers, known IP addresses and data contents. The problems with this technique are:

1. It relies on being able to inspect the inner packet contents for meaningful information, which is not possible if a particular IP flow is encrypted for example
2. Well known port numbers are just that, well known, and can be changed very easily. This technique assumes that a particular port number is always used for a particular service and will assume any flows to/from the well known port number are for the well known application, which is not always true
3. Application layer protocols regularly change and cannot be relied upon as an accurate source of information
4. Implementations using this technique require frequent external updates to keep the rule list mapping packet characteristics to traffic type up to date and reliable.

TCAPS can be considered to be a superset of Ubicom's StreamEngine technology, with some major advantages. TCAPS is able to centrally control and dynamically update customised QoS rules for each piece of CPE, whereas StreamEngine needs to be given a complete external rule set from the manufacturer in order to provide new or updated QoS rules and functionality. TCAPS can also classify and prioritise realtime/interactive traffic that is encrypted or running on non standard port numbers. If Ubicom's StreamEngine technology could be modified to accept signalling commands from the TCAPS manager, it could become part of the TCAPS solution.

3.5 High end networking equipment manufacturers

High end networking equipment manufacturers such as Cisco Systems Inc. and Alcatel Communications have integrated "automated" QoS features into their high end switches and routers. The term "automated" is slightly misleading in this sense. What these features allow is users to prepare one set of QoS "rules" on one device that will then distribute the same rule set to all devices on the network that are of the same type and from the same manufacturer. The system still requires user intervention to define/update the QoS policies/rules manually.

Allot Communications Ltd. have a family of products called NetEnforcer [10] on the market for IP carriers and service providers, which aim to provide simplified management of bandwidth control and service level management. The same problems inherent in the networking equipment "automated QoS" solution discussed above apply to the NetEnforcer product range. Applications requiring prioritisation on a link managed by NetEnforcer need to be provisioned manually before they are able to receive the QoS they require.

TCAPS is able to automatically create and distribute customised rules to each piece of CPE based on the traffic flowing to and from the CPE. No user intervention is required to define which flows contain realtime/interactive traffic or to create, update or remove the rules that control QoS being given to these flows.

4. TCAPS Architecture

The TCAPS architecture has evolved since its initial inception into a robust, scalable and efficient architecture for automated QoS provision over consumer broadband links. The following sub-sections will recap some of the system's guiding principles and design considerations, followed by a look at the resulting architecture, its features and evolution.

4.1 Guiding Principles & Design Goals

The high level guiding principles and design goals behind the current TCAPS architecture are outlined below:

- Module software should be compatible with industry standard platforms and environments where possible.
- TCAPS enable CPE should be backwards compatible with standard CPE i.e. a TCAPS enable CPE should fall back to functioning as a standard CPE when TCAPS network nodes are unavailable.
- The architecture should not affect non TCAPS related network operations/connectivity.
- The system should attempt to minimise the introduction of additional latency and jitter on non realtime/interactive traffic where possible, whilst ensuring that the focus remains on minimising latency and jitter for realtime/interactive traffic.
- The system should be able to classify packet flows even if the packet contents are encrypted.
- The architecture should be able to integrate an external security framework to provide additional system security.
- The architecture and implementation should adhere to any relevant existing standards where possible.
- The architecture should provide simple to manage scalability and redundancy by way of modularly designed components.

4.2 Architecture Components

The TCAPS architecture consists of 4 mandatory components and 1 optional component, all of which can be classified as either client side or ISP side.

The broadband access CPE device is the only mandatory client side TCAPS component, and resides at the customer premises. It is responsible for prioritising the transmission of realtime/interactive traffic onto the upstream link, as instructed to do so by the ISP side TCAPS Manager component.

The mandatory ISP side TCAPS components are the Manager, Flow Sifter and Flow Classifier, which all reside within the ISPs network. These components are all software applications, and should be thought of independently to the underlying hardware that is required to run them. They communicate with each other using UDP over IP, and therefore can be thought of independently of one another, as long as there is IP network connectivity between them.

The Manager component is responsible for managing the TCAPS enabled CPE. It performs administrative duties as well as signalling prioritisation rules to client CPE. It is the only ISP side component that communicates with CPE.

The Flow Sifter component is responsible for sifting client traffic into flows, calculating statistical features for each flow and sending these features to the Flow Classifier for classification.

The Flow Classifier component is responsible for classifying realtime/interactive client flows based on the statistical flow features received from the Flow Sifter. Flows identified as realtime/interactive are signalled to the Manager, which then instructs the client to prioritise the particular flow.

The ISP downstream QoS device is an optional ISP side TCAPS component. The ISP downstream QoS device does for the ISP to CPE downstream link what the TCAPS enabled CPE does for the CPE to ISP upstream link. That is, it prioritises realtime/interactive client traffic flows travelling downstream.

4.3 Architecture Configuration Options

With a basic understanding of the TCAPS components, let us examine the component configuration options available within the architecture.

The conventions used in the following diagrams are as follows:

- Nodes highlighted in red actively participate in the functioning of TCAPS
- Links highlighted in red carry TCAPS signalling information between the nodes
- An arrow on the end of a line indicates a unidirectional flow of network traffic in the direction pointed to by the arrow head

Figure 3 illustrates the simplest form of TCAPS deployment, building on the scenario presented in Figure 1.

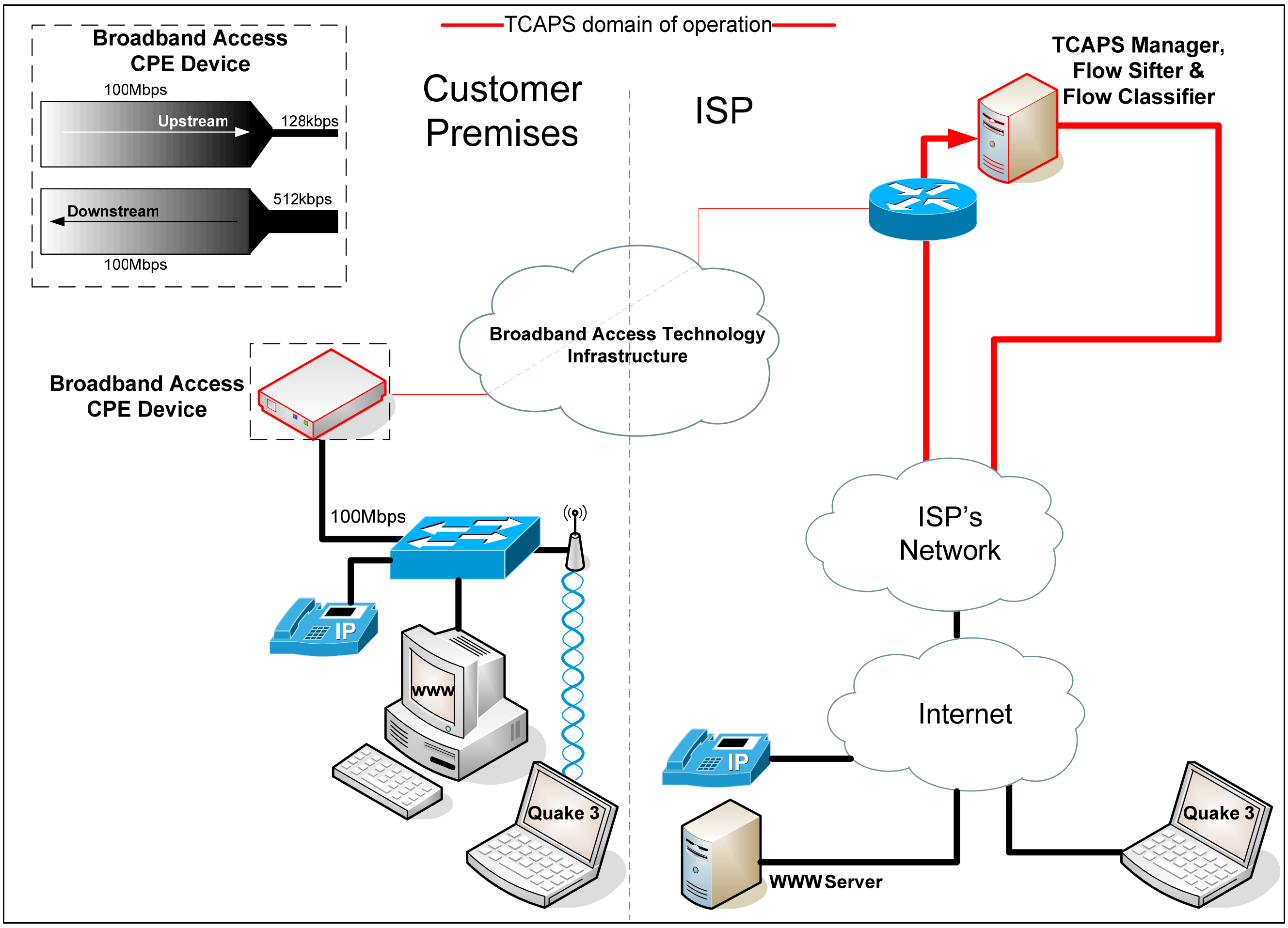
TCAPS domain of operation
Broadband Access CPE Device
100Mbps
Upstream
128kbps
512kbps
Downstream
100Mbps
Customer Premises
ISP
TCAPS Manager, Flow Sifter & Flow Classifier
Broadband Access Technology Infrastructure
Broadband Access CPE Device
100Mbps
IP
WWW
Quake 3
ISP's Network
Internet
IP
WWW Server
Quake 3

Figure 3. TCAPS architecture, simplest configuration

Figure 3 introduces a single additional server to the ISP's network, which groups all ISP side TCAPS components onto one physical machine. This machine is responsible for monitoring client flows, classifying realtime/interactive client flows and managing TCAPS enabled CPE. The unidirectional flow into the server from the router is a network tap of all CPE/ISP traffic. This tap is used by the flow sifter to perform its functionality. The network link between the server and ISP network cloud is used for all communications with the ISP network and CPE being managed by the server.

Figure 4 provides a view of Figure 3 with multiple clients. The client home network is now represented as a network cloud, and we can see that the single TCAPS server is able to concurrently manage all of these clients.

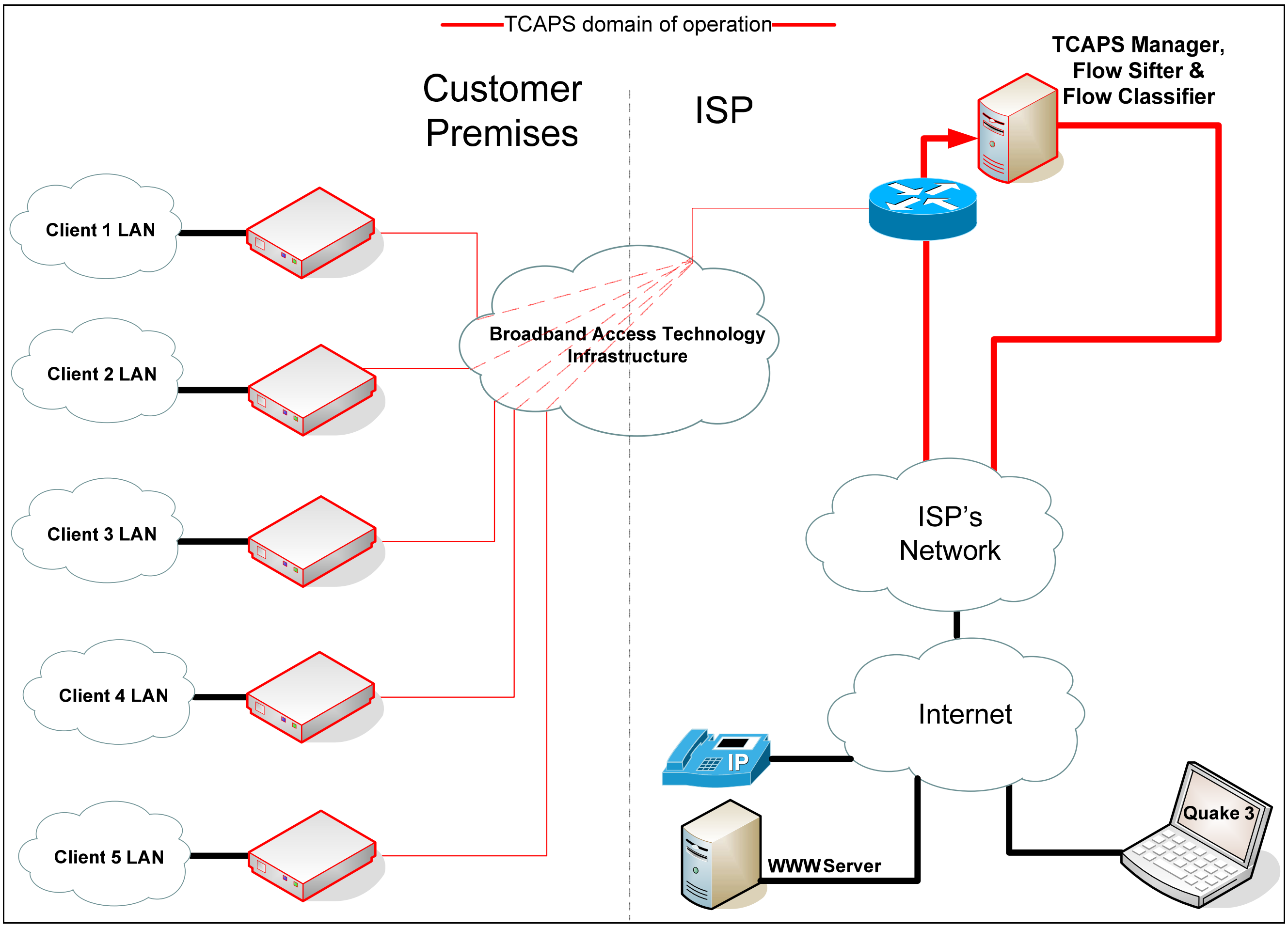

Figure 4. TCAPS architecture, simplest configuration with multiple clients

Figure 5 is beginning to show the flexibility provided by the networked ISP side software components.

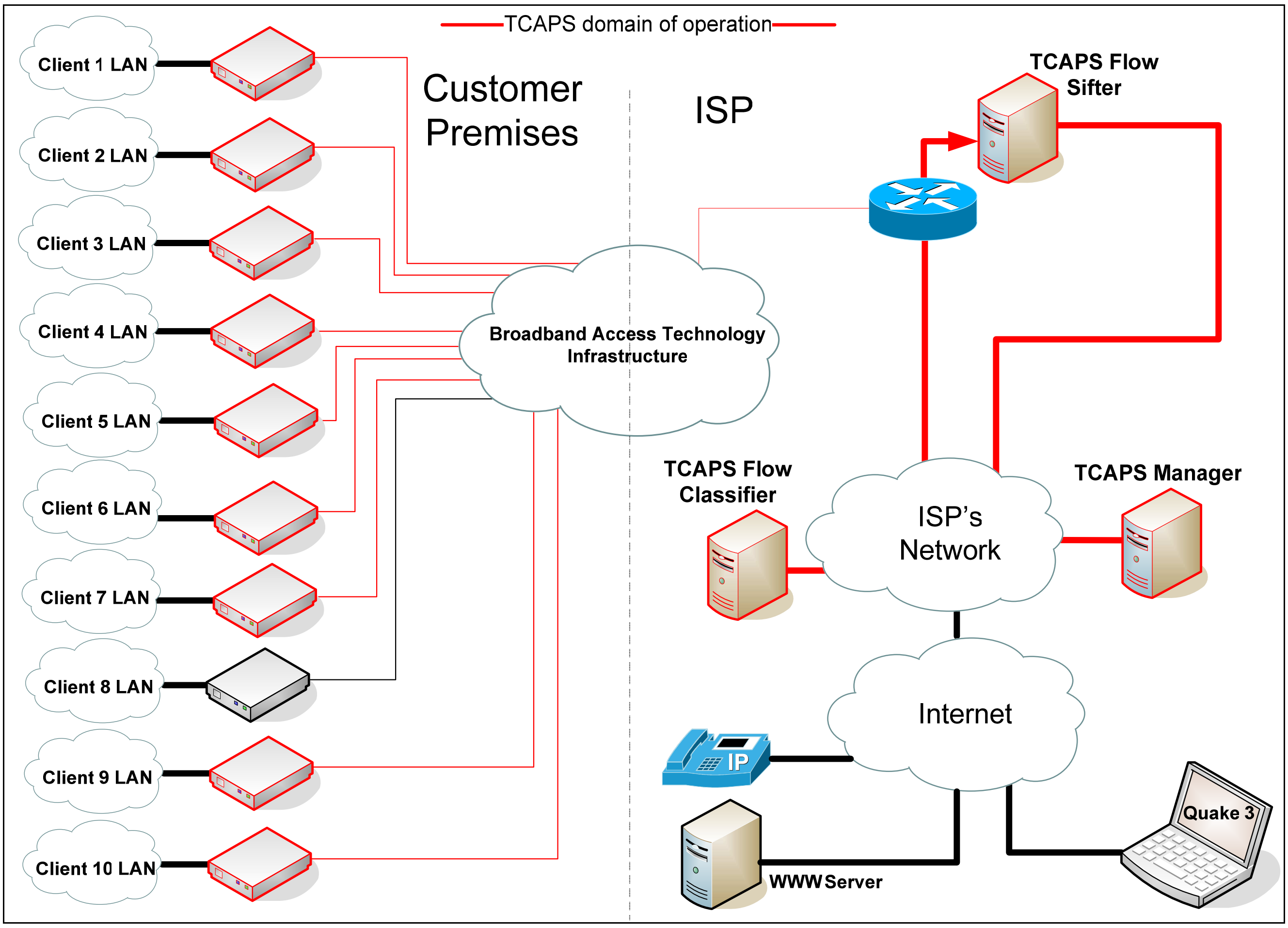

Figure 5. TCAPS architecture, moderately complex configuration with multiple clients

In order to accommodate more users, the individual components can be separated from each other and run on separate physical machines. The flow sifter requires a network tap of all CPE/ISP traffic, so it is slightly limited as to where it can be placed within the ISP network. However, the manager and flow classifier have no such restrictions, and simply need IP connectivity between themselves, the flow sifter and the client CPE devices.

The reason for "client 8" being black instead of red requires some explanation. The architecture is designed to allow standard CPE connecting to a TCAPS enabled ISP to function as they would in a non TCAPS enabled ISP. We can see that the path to the Internet for "client 8" is not reliant upon passing through a piece of TCAPS infrastructure and therefore perceives no difference in the network. This satisfies the TCAPS design goal relating to interoperability. An ISP deploying TCAPS in their network will not require all customers to replace their CPE with new TCAPS enabled devices. Rather, only customers wishing to take advantage of TCAPS will need to replace (or in some cases update) their existing modem.

It should also be noted that TCAPS enabled CPE differ from standard CPE in two areas that are not critical to the device's ability to function as a standard CPE. This means that TCAPS enabled CPE can function as regular CPE if the ISP they are connecting to is not TCAPS enabled, or if one or more of the ISP's mandatory TCAPS component servers has failed. This ensures there is a safe fall back mode in the event of no TCAPS service being provided, and means the user will still have network connectivity.

Figure 6 demonstrates the full flexibility and scalability of the TCAPS architecture.

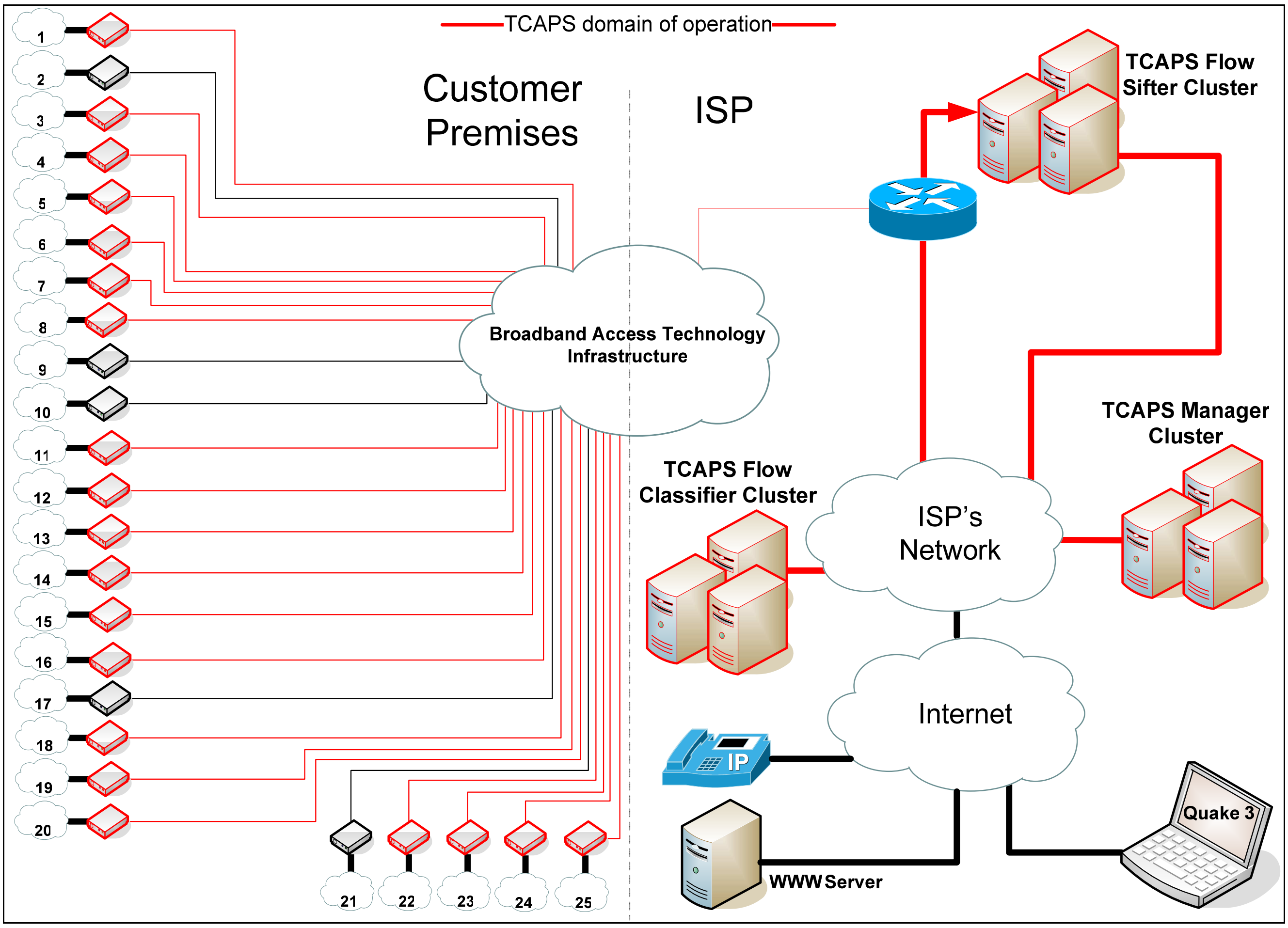

Figure 6. TCAPS architecture, complex configuration with multiple clients

When one physical server for each of the TCAPS components is insufficient to handle all of the ISP's clients, additional servers for each TCAPS component type can be added to form component clusters. This ability to scale hardware requirements linearly with demand ensures ISPs do not need to worry about complex capacity planning. When one of the component clusters appears to be working close to capacity, another physical machine can be added to the cluster to bring the overall workload down. The ability to cluster components also provides redundancy in the event that one of the cluster nodes fails.

It should be noted that while Figure 6 shows the cluster sizes for each of the TCAPS components to have grown by the same amount from Figure 5, this is not a requirement. For example, in the current prototype implementation, the flow sifter is the most hardware intensive application, while the flow classifier and manager are not overly intensive. We could therefore have grown the flow sifter cluster to 5 nodes and the flow classifier and manager clusters to only 2 nodes to make the most efficient use of our hardware resources.

The redundancy, scalability and ability to efficiently deploy hardware resources where they are most required satisfy a number of the overall TCAPS design goals.

As a final capability of the TCAPS architecture, Figure 7 illustrates the addition of the optional ISP downstream QoS device. It should be noted that this device does not interfere with the traffic of non TCAPS enabled CPE, thus retaining the interoperability discussed in the previous examples.

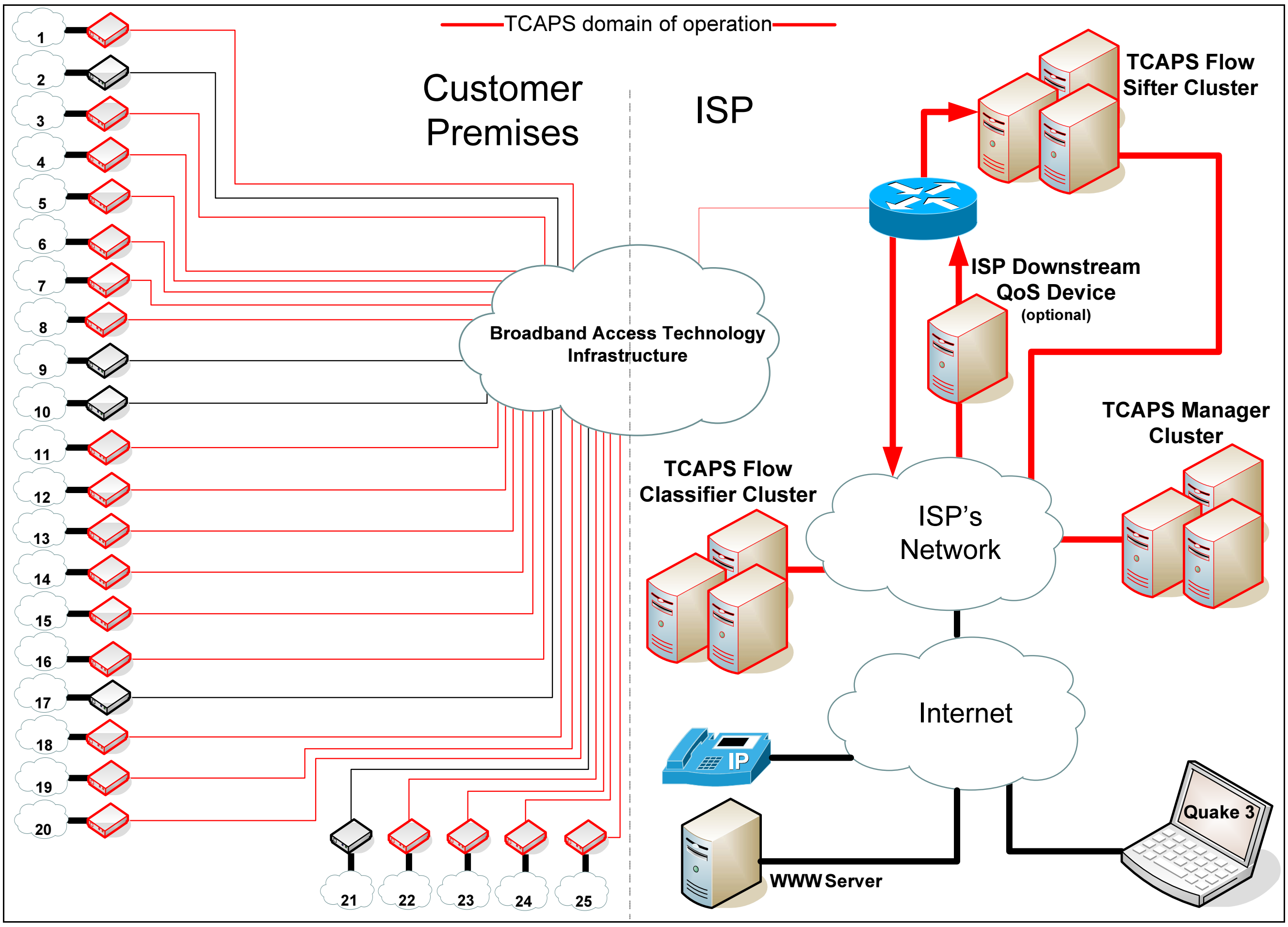
TCAPS domain of operation
Customer Premises
ISP
TCAPS Flow Sifter Cluster
ISP Downstream QoS Device
(optional)
Broadband Access Technology Infrastructure
TCAPS Manager Cluster
TCAPS Flow Classifier Cluster
ISP's Network
Internet
IP
Quake 3
WWW Server
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

Figure 7. TCAPS architecture, complex configuration with multiple clients and ISP downstream QoS device

Whilst Figure 7 shows the ISP downstream QoS device as an additional machine in the path of ISP/Internet traffic travelling towards the CPE, it is anticipated that this device's functionality would be build into the ISP's client side router in a commercial implementation.

However, queuing at the ISP for the downstream link is typically not as much of a problem as it as at the CPE for the upstream link. This is because the downstream speeds tend to be much higher (typically at least 4 times greater) than upstream speeds for typical broadband access Internet plans. This means that packets are able to be sent downstream 4 times faster than they can upstream, and the serialization delay is 4 times shorter. The serialization delay for a 1500 byte packet being sent downstream at 512kbps from the ISP is 23.43ms. Even if a packet from a realtime/interactive flow gets queued behind a 1500 byte packet, it will only have to wait 24ms instead of 94ms as before. This observation validates the need to particularly focus on a solution that reduces upstream realtime/interactive network traffic queuing delays and jitter. However, the downstream link would still benefit from the TCAPS solution in the same way as the upstream link would. Utilising TCAPS on the downstream link has therefore been made an optional part of the architecture. This is to cater for situations where the costs of implementing TCAPS on the downstream link may outweigh the benefits provided.

4.4 Features

The current TCAPS architecture provides a significant array of features aimed at simplifying the management of QoS provisioning and increasing the usability of realtime/interactive services over broadband network connections. The following list recaps the feature set provided by the TCAPS architecture:

- Redundancy through ability to cluster ISP side TCAPS components
- Scalability through ability to cluster ISP side TCAPS components
- Hardware resource optimisation through ability to grow component clusters independently of each other
- CPE to ISP upstream realtime/interactive traffic prioritisation
- Optional ISP to CPE downstream realtime/interactive traffic prioritisation
- Deployment flexibility through ability to group or split mandatory ISP side TCAPS components in any fashion on physical hardware
- Minimal requirements placed on TCAPS enabled CPE, as most of the hardware intensive work is done by the ISP side components
- Interoperability through ability for TCAPS enabled CPE and standard CPE to coexist within a TCAPS enabled ISP network
- Backwards compatibility through ability for TCAPS enabled CPE to function as standard CPE when ISP TCAPS components are not present or have failed
- Each client’s traffic is individually monitored and classified, which results in customised prioritisation rules being generated for each client
- Classification is performed based on the statistical features of flows rather than specific traffic characteristics, which provides the ability to classify encrypted flows

4.5 Deviations from initial design

The implementation phase of the TCAPS project has seen one significant change made to the overall TCAPS architecture, which will briefly be discussed and justified.

The original design of the ISP side of the TCAPS system incorporated the optional ISP downstream QoS device, and a device called the TCAPS server. The TCAPS server was functionally equivalent to the single TCAPS machine shown in Figure 4. The difference is that the software components in the original TCAPS server were not networked, unlike the components in the current architecture.

Like the current architecture, increasing the TCAPS client capacity in the previous design required the deployment of additional TCAPS servers. If each of the pieces of functionality provided by the TCAPS server (managing, flow sifting and flow classifying) were utilising as many resources as each other, then this solution would have been as efficient as the current architecture. However, some further research and experimentation during the early implementation stage showed that not all three pieces of functionality did use the same amount of resources. Flow sifting was found to consume significantly more resources than the other two pieces of functionality.

With this in mind, having followed through with the old design would have resulted in inefficient use of hardware resources. Flow sifting would have been limiting a server’s ability to handle more clients, rather than flow classification or client management. This would have resulted in additional servers being added which may not have been required had the individual pieces of functionality been running on separate hardware.

By modifying the architecture to its current state, the ISP administrator is able to group or split the three pieces of functionality depending on available hardware resources and client capacity requirements, and can assign better hardware to the functionality that requires the most resources.

Another minor benefit of this architecture change is the ability to move the flow classification and manager functionality to other places within the ISP's internal network (or even outside the ISPs network if required). This provides additional deployment flexibility to the ISP network administrator.

5. TCAPS Technical Overview

Implementing the TCAPS prototype has required a great deal of technical research, skill development and expansion of existing knowledge. The current TCAPS prototype has been implemented entirely in software for speed of development and deployment flexibility. The use of industry leading technologies, careful procedural/object oriented software design and advanced software programming features such as multithreading and raw packet sniffing ensure the software is robust, maintainable and scalable. The following sub-sections will recap some of the system's technical guiding principles and design considerations, followed by a look at the resulting components, their technical features and evolution.

5.1 Guiding Principles & Design Goals

The TCAPS technical guiding principles and design goals are a specialised subset of the TCAPS architectural guiding principles and design goals. They provide more stringent goals for the actual implementation of the system, and are outlined below:

- External software used in the system or software developed for the system should be cross platform where possible to allow maximum deployment flexibility.
- The system should make use of existing open source code where the release license permits and it is feasible to do so.
- Network overhead caused by the system should be kept below 0.5Mb per CPE/ISP link per day.
- The system should be able to classify IPsec encrypted packet flows.
- The system should be able to run on commodity and server grade hardware.
- The system should adhere to the existing standards for Ethernet, IPv4, IPv6, TCP and UDP where relevant.
- The system should be able to identify realtime/interactive flows within 20 seconds of the flow beginning.
- System must be able to support 10Mbit line rate traffic and should be able to handle 100Mbit line rate traffic running on commodity hardware.

5.2 Supporting technologies

A number of supporting technologies have been used in the development of the TCAPS prototype system. They provided mature solutions to some of the requirements of various TCAPS modules, and were incorporated where possible to reduce development time and effort.

5.2.1 Priority Queuing

Priority queuing was the technology selected to provide prioritisation functionality in both the broadband access CPE and optional ISP downstream QoS device. Priority queuing is a widely implemented QoS mechanism in many commercial and open source QoS systems.

Figure 8 illustrates the operational logic of a simple two queue priority queuing subsystem.

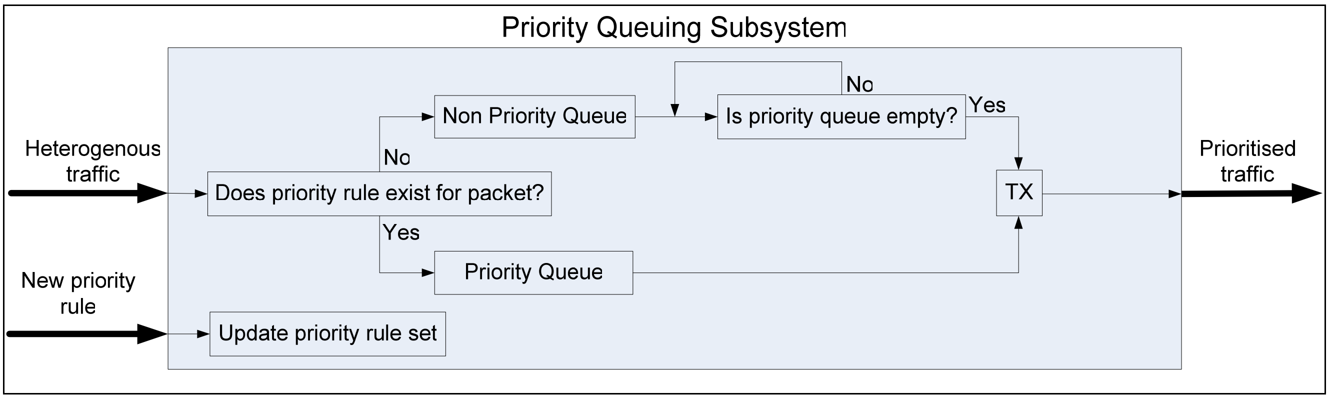

Figure 8. Priority queuing subsystem operation logic

Figure 9 and Figure 10 illustrate the same example discussed earlier in Figure 2, but with a priority queue (in red) for realtime/interactive traffic, and another queue (in black) for other traffic.

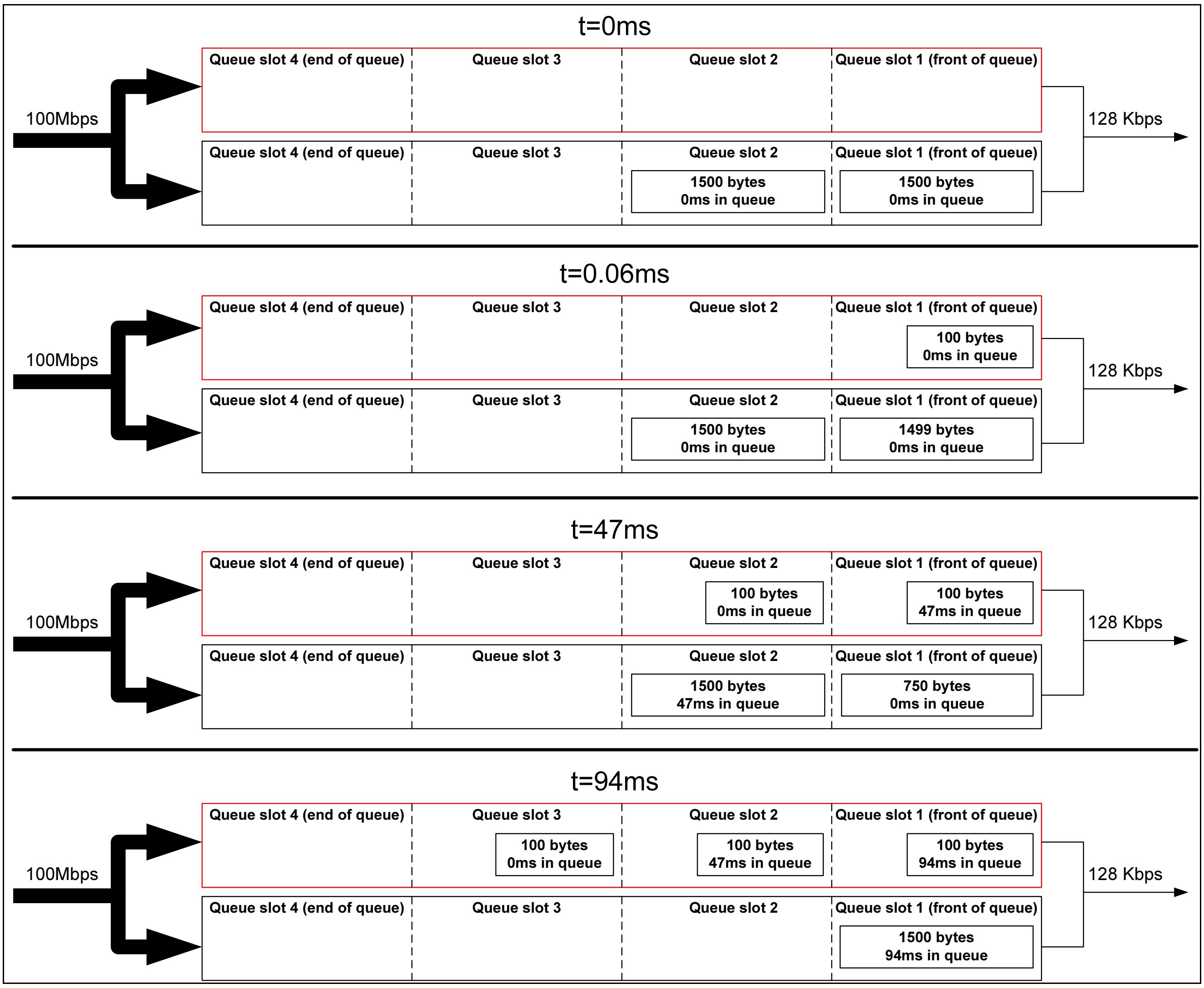

Figure 9. Time sequence diagram for CPE with 2 upstream queues, 0 – 94ms

The priority queuing system behaves exactly the same as the normal queuing system up to t=94ms. This is because once a packet begins being serialized onto the upstream link, it cannot be stopped. Therefore, the realtime/interactive packet that arrives just after the 1500 byte packet at t=0.06ms has to wait until the 1500 byte packet is transmitted, which finishes at t=94ms.

The affect of the priority queuing is visible in the time after t=94ms, which is shown in Figure 10.

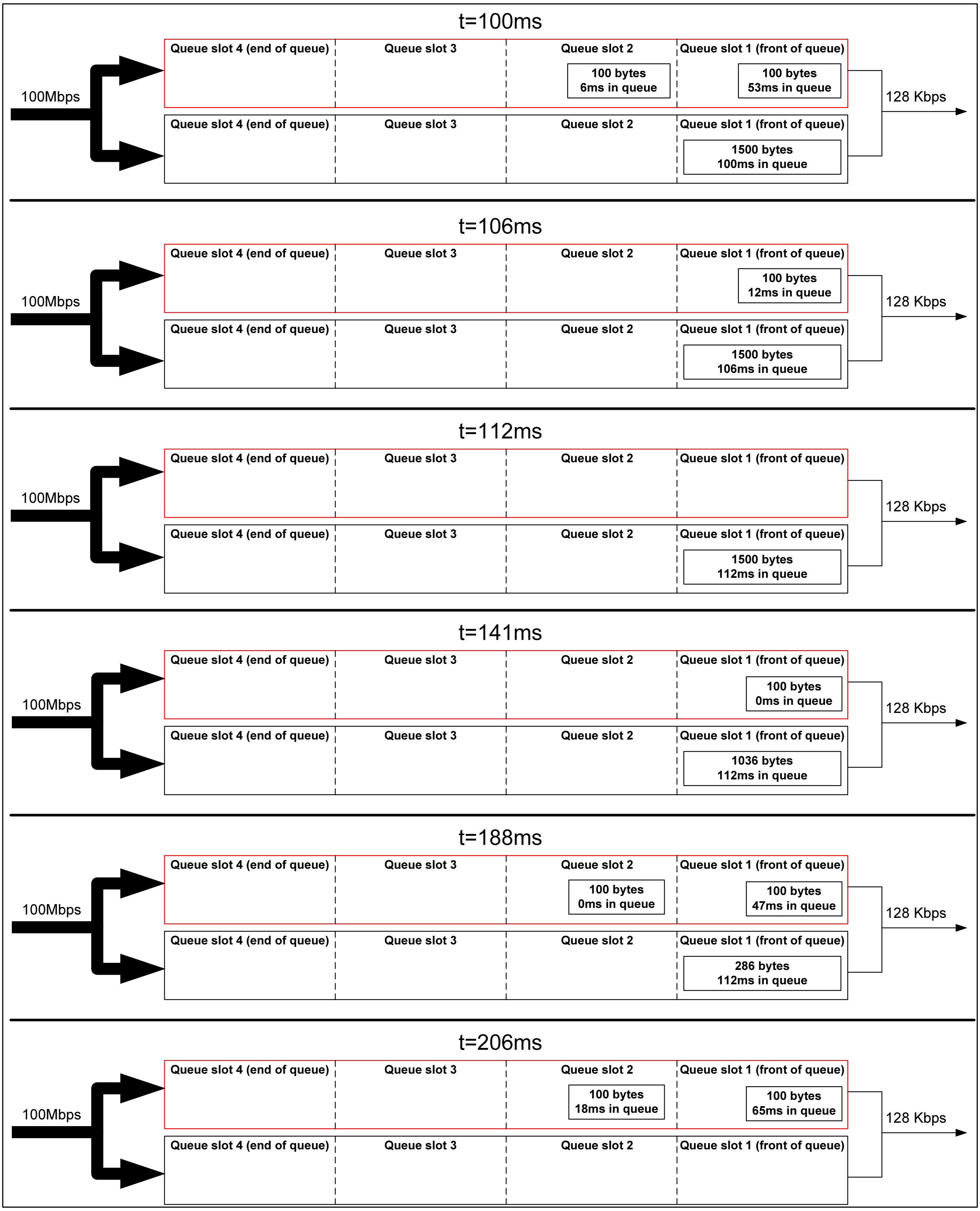

Figure 10. Time sequence diagram for CPE with 2 upstream queues, 100 – 206ms

We can see at t=100ms that the first 100 byte packet has been sent, as in the Figure 2 example. However, instead of the second 1500 byte packet beginning transmission, the second 100 byte packet waiting in the priority queue is transmitted instead, followed by the third 100 byte packet that was waiting behind the second. At t=112ms, the priority queue is finally empty and the upstream link is clear for the second 1500 byte packet to begin transmission. The remainder of the diagram shows two more 100 byte packets arriving a the priority queue and having to wait for the second 1500 byte packet to finish transmission.

The end result of this scenario is that a traffic flow that should have consistent 47ms inter-packet arrival times (when not subjected to queuing delays), experiences inter-packet arrival times of 141ms, 100ms, 59ms, 112ms and 71ms. If we compare these times with the normal queuing example times of 141ms, 194ms, 153ms, 112ms and 71ms seen in Figure 2, we see same or smaller queuing times in the priority queuing example.

It is certainly not a perfect solution, but the most important aspect of the results is that priority queuing constrains the maximum queue wait to the serialization delay of the largest packet that could end up in the queue. For example, if an additional two 1500 byte packets had entered the non-priority queue at t=0ms, the delay times for the last two 100 byte packets at t=206ms would remain unchanged. This comes as a result of them being given priority on the upstream link at t=206, regardless whether anything exists in the non-priority queue.

5.2.2 The FreeBSD UNIX Operating System

The FreeBSD UNIX operating system [11] is an open source operating system released under the revised BSD licence. It provides a comprehensive array of features and applications useful for providing any sort of network service platform, making it an ideal choice as the development platform for the TCAPS system prototype.

5.2.3 The PF/ALTQ Firewall/Queuing Subsystem

The PF/ALTQ [12] [13] firewall and queuing subsystem is an open source, *BSD UNIX based system for IP packet firewalling and queuing. It provides a priority queuing implementation and comes included in the FreeBSD operating system, making it an ideal choice for the QoS subsystem used within the prototype broadband access CPE.

5.2.4 The PCAP Library

The PCAP library (libpcap) [14] is an open source software library written in the C programming language, released under the revised BSD licence. It provides a cross platform API for sniffing raw packets from a network interface, independent of the more traditional operating system network sockets API.

libpcap has been used in the TCAPS flow sifter module to provide raw (OSI layer 2) packet sniffing capabilities to the software. libpcap allows the flow sifter to sniff packets from a network interface that has no outgoing communications capabilities i.e. a network tap. This satisfies the TCAPS design goal relating to minimal impact, as the flow sifter does not need to be placed in the CPE/ISP communications path. Instead, the CPE/ISP communications path can remain unchanged, and a simple tap of the traffic travelling over the path can be provided to the flow sifter.

The fact that libpcap runs on many platforms, including Microsoft Windows, Linux, *BSD UNIX and Mac OS X, means the flow sifter is not limited to running on a specific operating system. This satisfies the TCAPS design goal relating to maximising cross platform compatibility.

5.2.5 The ACE Library

The Adaptive Communications Environment library (libACE) [15] is an open source software library written in the C++ programming language, released under a licence very similar to the revised BSD licence. It provides a comprehensive, cross platform solution to developing high-performance and real-time communication services and applications[1]. Figure 11 provides an overview of the components and frameworks provided by the ACE.

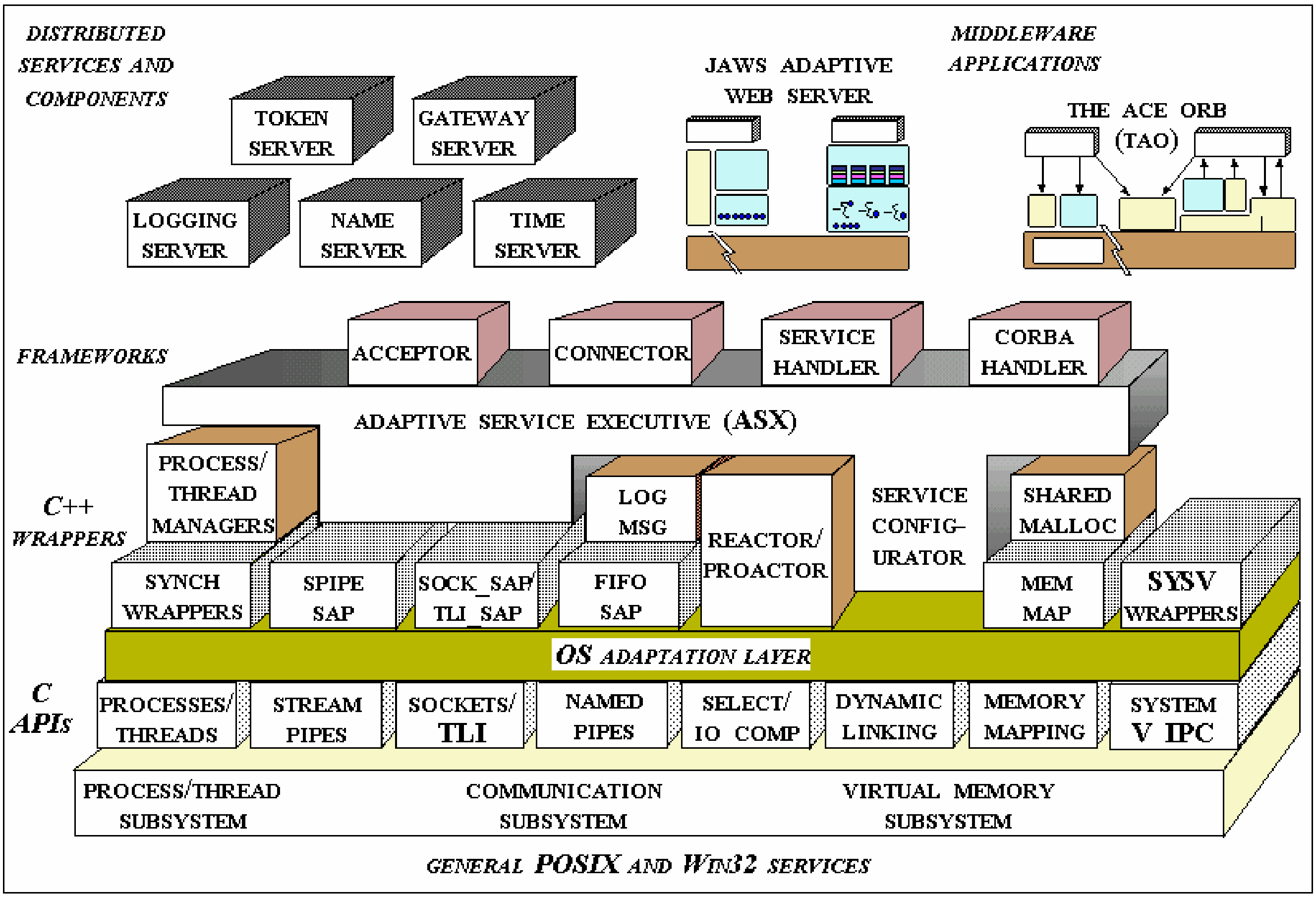

Figure 11. ACE overview (taken from http://www.cs.wustl.edu/~schmidt/ACE-overview.html)

libACE has been used extensively in all of the C++ applications developed as part of the TCAPS prototype. It has been primarily used to provide threading and sockets related functionality to the applications, and has significantly reduced the development time for these aspects of the TCAPS software.

Using libACE not only simplifies the amount of development work required to build a C++ network application, but also allows the software to be run on any platform ACE runs on. The list of supported platforms is quite comprehensive, with Microsoft Windows, Mac OS X, Linux and *BSD UNIX all supported. The use of libACE by TCAPS therefore affords ISPs a great deal of flexibility in terms of the choice of platform for their TCAPS deployment. This satisfies the TCAPS design goal relating to maximising cross platform compatibility.

---

[1] http://www.cs.wustl.edu/~schmidt/ACE-overview.html

5.3 Deviations from initial design

The implementation phase of the TCAPS project has seen a few changes made to some of the TCAPS component designs, which will briefly be discussed and justified.

As a result of the decision to change the TCAPS architecture early in the implementation stage, the TCAPS signalling protocol module had to be extended to support three new communication types between the new networked entities.

The client management packet type was implemented to facilitate communications between the manager and flow sifter modules. Managers use this packet type to tell flow sifters the IP address of clients they must observe traffic for.

The flow feature packet type was implemented to facilitate communications between flow sifters and flow classifiers. Flow sifters use this packet type to send their calculated flow features for each client flow to flow classifiers.

The classifier rule management packet was implemented to facilitate communications between flow classifiers and managers. Flow classifiers use this packet type to tell managers about detected realtime/interactive flows that belong to clients under their control.

Some of the existing signalling protocol packet definitions also had to be slightly modified from their initial specification, to account for unforseen implementation issues. This mostly consisted of adding fields used to hold the length of other variable length fields within the packet.

Finally, owing to a lack of knowledge about writing networked applications, the original design choice to use libpcap/libnet for packet sending/receiving was extremely misguided. These libraries are useful for uni directional packet sniffing and sending respectively. They are not overly suitable to being used for the sending and receiving of network communications between two applications. As a result of this, the TCAPS prototype implementation utilised UDP sockets for application network communications. Sockets are the standard way of performing such communications and are much simpler to use for these tasks than the libpcap/libnet libraries.

6. TCAPS Detailed Technical Discussion

6.1 Signalling Protocol

The TCAPS signalling protocol module provides the glue between all of the TCAPS application modules. It facilitates all communications within the TCAPS framework via 7 different packet types: the rule management packet, classifier rule management packet, acknowledgment packet, transfer packet, poll packet, flow feature packet and the client management packet.

The TCAPS signalling protocol module has been implemented as a procedural C library. This design choice ensured that if any external parties were to become involved in the commercialisation of TCAPS, the protocol code base would most likely be compatible with their development platforms. This is particularly pertinent to CPE devices, which tend to run on embedded hardware platforms. Such platforms generally have C compilers readily available, but lack other, high-level language development platforms.

The protocol has been implemented as a binary protocol. This design choice simplified the structure of the protocol itself, and ensured minimal overhead of packet sizes on the wire. Minimal overhead was an important consideration for the protocol, as it reduces the negative impact of TCAPS on the network. Too much signalling protocol overhead would have reduced the benefits provided by TCAPS.

The protocol has been designed for use with both connectionless and connection oriented transport mechanisms. By adding an acknowledgment packet type to the protocol, the TCAPS applications can handle reliable end to end signalling transport, rather than relying on a connection oriented transport mechanism. The TCAPS prototype presented in this report has made exclusive use of UDP over IP as the signalling protocol transport mechanism. This design choice further minimised the overhead of TCAPS signalling traffic on the network. UDP does not require an established connection between end hosts for data transfer, and does not require acknowledgments of each packet, compared to TCP which uses both. However, there is nothing stopping the use of the protocol over TCP either. As such, network operators and equipment manufacturers have ultimate flexibility in the way they choose to utilise the protocol.

The following 7 sub-sections examine each of the individual packet types and their uses in more detail. The “Field Type” table column referred to in each of the sub-sections refers to the C programming language data type used to hold the value for that field. U8 refers to an unsigned 8 bit type, U16 refers to an unsigned 16 bit type, U32 refers to an unsigned 32 bit type, U8 * refers to a string of bytes (variable or fixed length) and float refers to a 32 bit floating point number type.

Note that values for fields such as “Packet Type”, “Match Rule” and “Poll Type” were arbitrarily defined during implementation. The correct mapping of meaning to value for such fields can always be found in the tcaps_sp.h library header file.

6.1.1 Rule Management Packet Type

The rule management packet (RMP) type is used to communicate QoS SS related rules in a QoS SS independent fashion. It is primarily used between the TCAPS manager and TCAPS client interface, but can be used between TCAPS managers during transfer operations.

The serialized structure of a RMP is shown in Figure 12, and the serialized structure of a rule segment is shown in Figure 26.

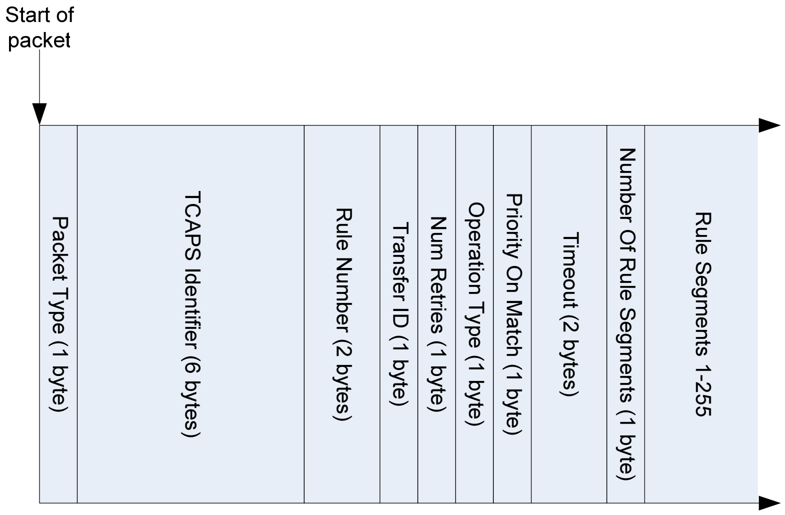

Figure 12. Rule management packet serialized view

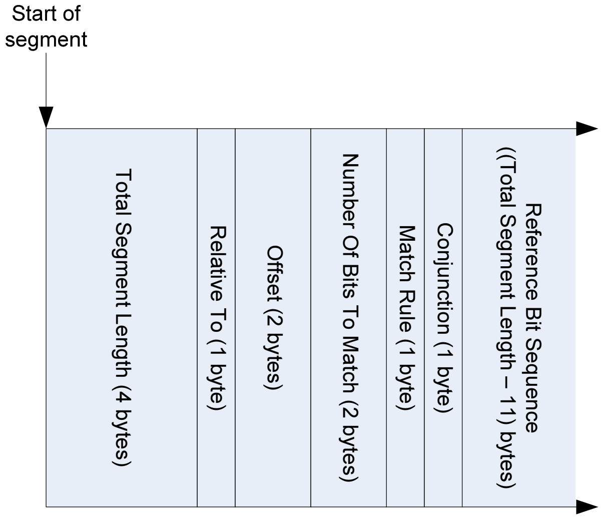
Start of segment
Total Segment Length (4 bytes)
Relative To (1 byte)
Offset (2 bytes)
Number Of Bits To Match (2 bytes)
Match Rule (1 byte)
Conjunction (1 byte)
Reference Bit Sequence ((Total Segment Length – 11) bytes)

Figure 13. Rule segment serialized view

Table 1 summarises each of the RMP fields.

Field Name	Field Length (bytes)	Field Type	Field Description
Packet Type	1	U8	Identifies the packet as a RMP
TCAPS Identifier	6	U8 *	MAC address of the client device. Used to uniquely identify the client to the TCAPS manager.
Rule Number	2	U16	Uniquely identifies the RMP and associated rule
Transfer ID	1	U8	Identifies the transfer operation, if any, this RMP is being sent as part of
Num Retries	1	U8	Identifies the number of times this particular RMP has been sent
Operation Type	1	U8	Identifies whether the associated rule is to be created, updated or deleted
Priority On Match	1	U8	Identifies the priority that should be given to traffic matching the associated rule. 0 is highest priority, 255 is lowest priority

Field Name	Field Length (bytes)	Field Type	Field Description
Timeout	2	U16	The amount of time (in seconds) after rule instantiation that the rule will expire. A timeout of 0 means rule never expires
Number Of Rule Segments	1	U8	Defines how many rule segments make up the associated rule. Must be in range [1,255] to be valid

Table 1. Rule management packet fields

Table 2 summarises each of the rule segment fields.

Field Name	Field Length (bytes)	Field Type	Field Description
Total Segment Length	4	U32	Total length (in bytes) of the segment, inclusive of this 4 byte field
Relative To	1	U8	Specifies which well known packet entry point the rule segment is relative to e.g. beginning of IP header, beginning of TCP header, etc.
Offset	2	U16	Specifies the offset (in bits) into the packet from the "Relative To" packet entry point
Number Of Bits To Match	1	U8	Specifies the number of bits from "Offset" to match with the reference bit sequence
Match Rule	1	U8	Specifies the type of match to apply between the packet bit sequence and reference bit sequence e.g. equal to, greater than, etc.
Conjunction	1	U8	Specifies the relationship between this rule segment and the next e.g. "and", etc.
Reference Bit Sequence	"Total Segment Length" -11	U8 *	The bit sequence to compare the packet's bit sequence to

Table 2. Rule segment fields

A RMP is constructed by filling in the appropriate fields in the base packet, and then appending between 1 and 255 rule segments to the end of the base packet. Each rule segment contains the information required to build a portion of a complete rule. The segments "string" together using the conjunction field of each segment, and effectively form a Boolean expression that a packet will either match or not match. Packets that match the rule will be given the level of priority indicated in the "Priority On Match" field.

For example, let us construct a rule for a client with MAC address 12:34:56:78:9A:BC that gives the highest possible priority to all TCP traffic travelling between source 172.16.251.50:5000 and destination 136.186.229.95:43519 and expires in 5 minutes. We shall assume this is rule number 1 and this rule is not part of a transfer operation. Table 3 shows the 72 byte RMP corresponding to our example rule.

	Field Name	Field Value (Hex)	Field Value (Decimal)
Base RMP	Packet Type	0x01	1
	TCAPS Identifier	0x123456789ABC	20015998343868
	Rule Number	0x01	1
	Transfer ID	0x00	0
	Num Retries	0x00	0
	Operation Type	0x01	1

	Priority On Match	0x00	0
	Timeout	0x12C	300
	Number Of Rule Segments	0x04	4
Rule Segment 1	Total Segment Length	0x0F	15
	Relative To	0x02	2
	Offset	0x60	96
	Number Of Bits To Match	0x20	32
	Match Rule	0x01	1
	Conjunction	0x01	1
	Reference Bit Sequence	0xAC10FB32	2886794034
Rule Segment 2	Total Segment Length	0x0D	13
	Relative To	0x03	3
	Offset	0x00	0
	Number Of Bits To Match	0x10	16
	Match Rule	0x01	1
	Conjunction	0x01	1
	Reference Bit Sequence	0x1388	5000
Rule Segment 3	Total Segment Length	0x0F	15
	Relative To	0x02	2
	Offset	0x80	128
	Number Of Bits To Match	0x20	32
	Match Rule	0x01	1
	Conjunction	0x01	1
	Reference Bit Sequence	0x88BAE55F	2293949791
Rule Segment 4	Total Segment Length	0x0D	13
	Relative To	0x03	3
	Offset	0x10	16
	Number Of Bits To Match	0x10	16
	Match Rule	0x01	1
	Conjunction	0x01	1
	Reference Bit Sequence	0xA9FF	43519

Table 3. Example rule management packet

We shall briefly discuss the meaning of some of the less obvious field values in rule segment 1 and relate them back to our original example rule. Rule segment 1 is matching the source IP address of our rule. "Relative To" having a value of 2 refers to the start of a packet's IP header. To reach the source IP address relative to the start of the IP header, we have to move 96 bits into the header - hence the "Offset" of 96. Once at the beginning of the source IP field of the IP header, we need to match the next 32 bits to match an IP address. A "Match Rule" of 1 is defined as being "equal to" i.e. we want the full 32 bits of a packet's IP header source IP field to match all 32 bits of our "Reference Bit Sequence". A "Conjunction" of 1 is defined to be "and" i.e. in order for a packet to match our rule as a whole, it must match both this rule segment *and* the next rule segment. Finally, the "Reference Bit Sequence of 0xAC10FB32 is the binary equivalent of 172.16.251.50.

After analysing the remaining 3 rule segments in a similar fashion, we can show that the 4 rule segments combine to say "a packet will match this rule if its source IP address is 172.16.251.50 *and* its source port is 5000 *and* its destination IP address is 136.186.229.95 *and* its destination port is 43519". Represented in PF firewall parlance, this rule can be represented as "pass out on <interface_name> inet proto tcp 172.16.251.50 port 5000 to 136.186.229.95 port 43519".

One additional note on the ordering of rule segments: the current TCAPS client interface (which is the only module that translates RMPs into QoS SS specific rules) correctly constructs QoS SS specific rules from a RMP with any ordering of rule segments.

6.1.2 Classifier Rule Management Packet

The classifier rule management packet (CRMP) type is used to communicate QoS SS related rules for a particular client in a QoS SS independent fashion. It is sent from the TCAPS flow classifier to the TCAPS manager.

The serialized structure of a CRMP is shown in Figure 14.

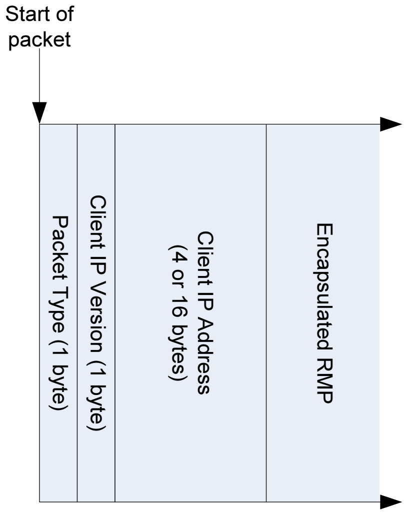

Figure 14. Classifier rule management packet serialized view

Table 4 summarises each of the CRMP fields.

Field Name	Field Length (bytes)	Field Type	Field Description
Packet Type	1	U8	Identifies the packet as a CRMP
Client IP Version	1	U8	Specifies the IP version of the “Client IP Address” field. A value of 1 corresponds to IPv4 and 2 corresponds to IPv6
Client IP Address	4 or 16	U8 *	Specifies the IP address of the client the encapsulated RMP is to be sent to. 4 bytes if “Client IP Version” is 1, 16 bytes if “Client IP Version” is 2
Encapsulated RMP	1	U8 *	A partially filled in RMP, with the “Packet Type”, “Priority On Match”, “Operation Type”, “Number Of Rule Segments” fields correctly filled in and the correct rule segments attached

Table 4. Classifier rule management packet fields

A CRMP is sent when a TCAPS flow classifier has identified a client traffic flow that requires a change in its current priority level. The CRMP type is effectively a RMP with an additional header, which identifies the particular client, by IP address, that the RMP is to be sent to. The TCAPS manager is the only module that deals directly with clients. Therefore, a TCAPS flow classifier must notify the TCAPS manager responsible for a particular client when a client rule needs to change, and the CRMP is specifically used for this purpose.

CRMPs will be sent from a TCAPS flow classifier to TCAPS manager if:

- A currently non prioritised flow (priority = 255) is assigned a priority less than 255
- A currently prioritised flow (priority < 255) is assigned a priority of 255
- A currently prioritised flow (priority < 255) is assigned a different priority less than 255

For a given client and a given flow, a change in the priority assigned to the client's flow at the TCAPS flow classifier will only result in 2 changes to the encapsulated RMP.

If a currently non prioritised flow (priority = 255) is assigned a priority less than 255, the RMP "Priority On Match" field will be set to the assigned priority, and the "Operation Type" field will be set to 1 to indicate the client should create the rule specified by this RMP.

If a currently prioritised flow (priority < 255) is assigned a priority of 255, the RMP "Priority On Match" field will be set to 255, and the "Operation Type" field will be set to 2 to indicate the client should delete the rule specified by this RMP.

If a currently prioritised flow (priority < 255) is assigned a different priority less than 255, the RMP "Priority On Match" field will be set to the assigned priority, and the "Operation Type" field will be set to 3 to indicate the client should update the rule specified by this RMP.

6.1.3 Acknowledgment Packet Type

The acknowledgment packet (AP) type is used to acknowledge the receipt of other signalling protocol packet types. All TCAPS modules can use the AP type.

The serialized structure of an AP is shown in Figure 15.

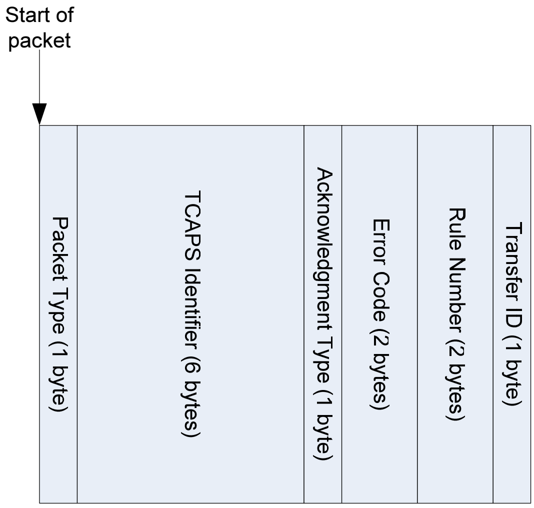

Figure 15. Acknowledgment packet serialized view

Table 5 summarises each of the AP fields.

Field Name	Field Length (bytes)	Field Type	Field Description
Packet Type	1	U8	Identifies the packet as an AP

Field Name	Field Length (bytes)	Field Type	Field Description
TCAPS Identifier	6	U8	MAC address of the sending device. Used to uniquely identify the source device to the recipient device
Acknowledgment Type	1	U8	Identifies the packet type being acknowledged
Error Code	2	U16	If the acknowledged packet caused an error, this field will specify a numeric code representing the error
Rule Number	2	U16	If the acknowledged packet is a RMP, this field will specify rule number of the RMP
Transfer ID	1	U8	If the acknowledged packet is part of a transfer operation, this field will specify transfer id of the transfer operation

Table 5. Acknowledgment packet fields

6.1.4 Transfer Packet Type

The transfer packet (TP) type is used to allow transfers of existing state between devices. It is primarily used between the TCAPS manager and TCAPS client interface, but can be used between TCAPS managers as part of a client transfer operation.

The different serialized structures of a TP are shown in Figure 16 and Figure 17.

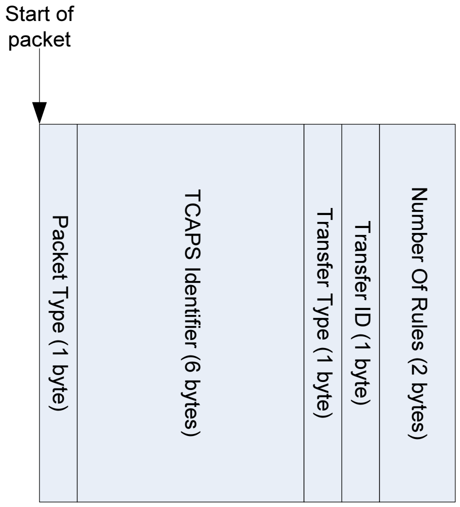

Figure 16. Transfer packet serialized view, "Transfer Type" is of type "Rule"

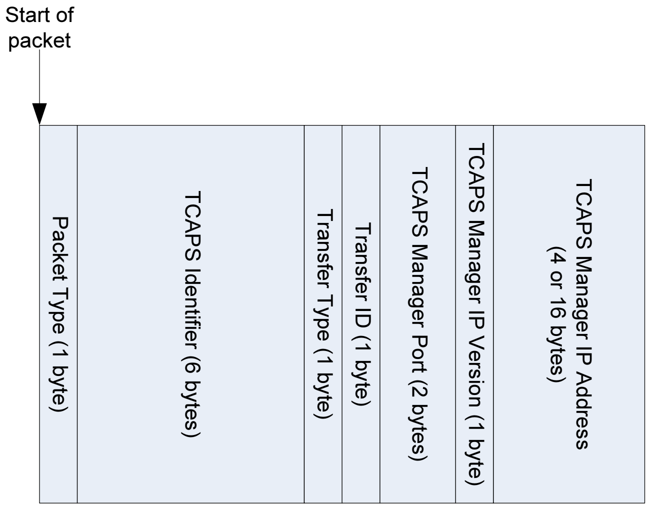
Start of packet
Packet Type (1 byte)
TCAPS Identifier (6 bytes)
Transfer Type (1 byte)
Transfer ID (1 byte)
TCAPS Manager Port (2 bytes)
TCAPS Manager IP Version (1 byte)
TCAPS Manager IP Address (4 or 16 bytes)

Figure 17. Transfer packet serialized view, "Transfer Type" is of type "Server"

Table 6 summarises each of the TP fields.

Field Name	Field Length (bytes)	Field Type	Field Description
Packet Type	1	U8	Identifies the packet as a TP
TCAPS Identifier	6	U8	MAC address of the sending device. Used to uniquely identify the source device to the recipient device
Transfer Type	1	U8	Identifies the requested transfer operation
Transfer ID	1	U8	A unique number (over the lifetime of the transfer operation) identifying the transfer operation
Number Of Rules	2	U16	If the "Transfer Type" is of type "Rule", this field will be present and will indicate the number of rules being transferred
TCAPS Manager Port	2	U16	If the "Transfer Type" field is of type "Server", this field will be present and specifies the UDP/TCP port number used by the TCAPS manager being transferred to
TCAPS Manager IP Version	1	U8	If the "Transfer Type" field is of type "Server", this field will be present and specifies the IP version of the "TCAPS Manager IP Address" field. A value of 1 corresponds to IPv4 and 2 corresponds to IPv6
TCAPS Manager IP Address	4 or 16	U8 *	If the "Transfer Type" field is of type "Server", this field will be present and specifies the IP address of the TCAPS manager being transferred to. 4 bytes if "TCAPS Manager IP Version" is 1, 16 bytes if "TCAPS Manager IP Version" is 2

Table 6. Transfer packet fields

6.1.5 Poll Packet Type

The poll packet (PP) type is used as a keep alive signal, as well as to perform client registration/deregistration. It is currently only used between the TCAPS manager and TCAPS client.

The serialized structure of a PP is shown in Figure 18, and the serialized structure of an information segment is shown in Figure 19.

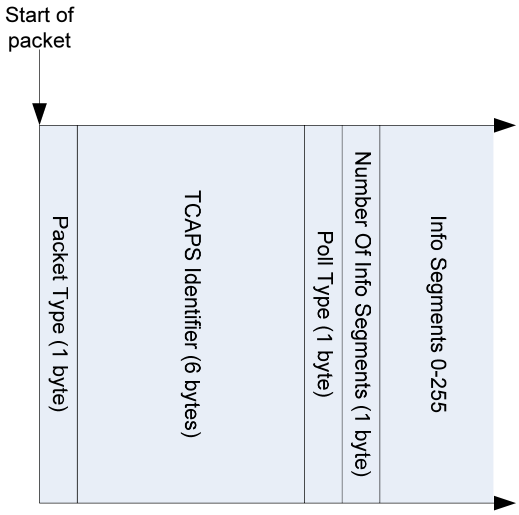

Figure 18. Poll packet serialized view

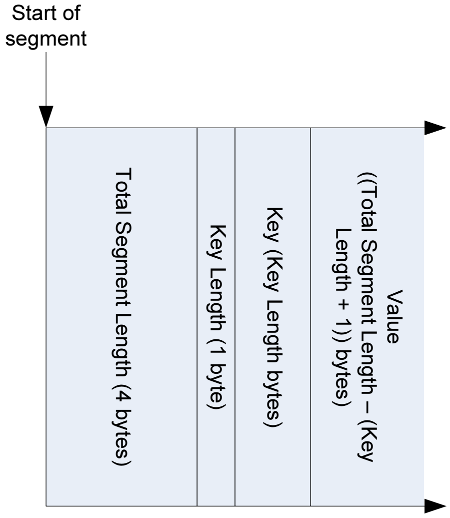
Start of segment
Total Segment Length (4 bytes)
Key Length (1 byte)
Key (Key Length bytes)
Value ((Total Segment Length – (Key Length + 1)) bytes)

Figure 19. Information segment serialized view

Table 7 summarises each of the PP fields.

Field Name	Field Length (bytes)	Field Type	Field Description
Packet Type	1	U8	Identifies the packet as an PP
TCAPS Identifier	6	U8	MAC address of the sending device. Used to uniquely identify the source device to the recipient device
Poll Type	1	U8	Identifies the purpose of the PP
Number Of Info Segments	1	U8	Defines how many information segments have been appended to the base PP. Entire range [0,255] is valid

Table 7. Poll packet fields

Table 9 summarises each of the information segment fields.

Field Name	Field Length (bytes)	Field Type	Field Description
Total Segment Length	4	U32	Total length (in bytes) of the segment, inclusive of this 4 byte field
Key Length	1	U8	Specifies the length (in bytes) of the “Key” field
Key	“Key Length”	U8 *	Contains the key for this information segment

Field Name	Field Length (bytes)	Field Type	Field Description
Value	"Total Segment Length" – ("Key Length" + 1)	U8 *	Contains the value associated with the "Key" for this information segment

Table 8. Information segment fields

A PP is constructed by filling in the appropriate fields in the base packet, and then appending between 0 and 255 rule segments to the end of the base packet. In its simplest form, a PP will have no appended information segments, and can be used as a keep alive, or for registration/deregistration of a TCAPS client interface with a TCAPS manager.

Information segments can contain any data, but were primarily implemented to allow devices to communicate their capabilities with each other. They are structured in a key-value pair arrangement, with a maximum key length of 255 bytes and unrestricted value length. The information segment's specification allows it to be used with ASCII text and/or binary keys/values. Limiting the "Key Length" field to 1 byte allows the "Key" field to be populated with up to 255 ASCII characters, or 2^{255*8} different binary combinations. This limitation should still provide more than enough different key combinations to allow communication of a full set of device characteristics.

6.1.6 Flow Feature Packet

The flow feature packet (FFP) type is used to communicate client specific flow features. It is sent from the TCAPS flow sifter to the TCAPS flow classifier.

The different serialized structures of a FFP are shown in Figure 20 and Figure 21, and the serialized structure of a flow feature segment is shown in Figure 22.

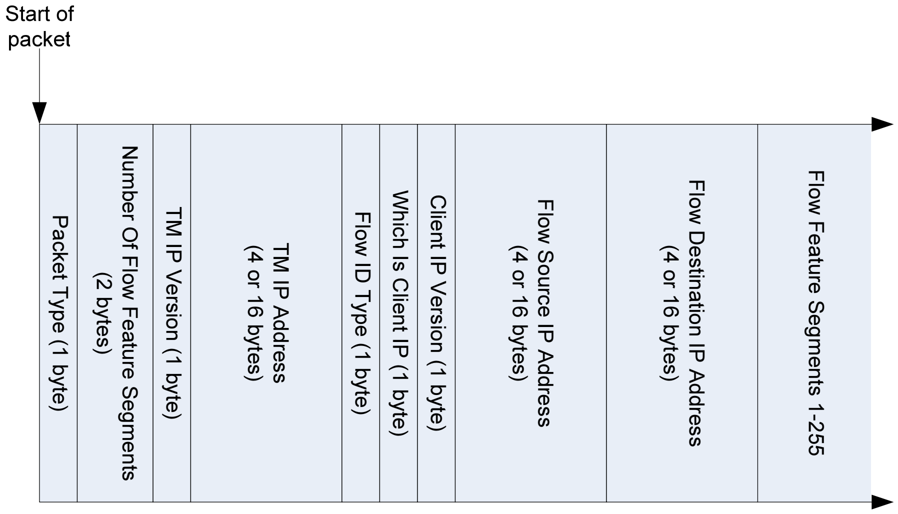
Start of packet
Packet Type (1 byte)
Number Of Flow Feature Segments (2 bytes)
TM IP Version (1 byte)
TM IP Address (4 or 16 bytes)
Flow ID Type (1 byte)
Which Is Client IP (1 byte)
Client IP Version (1 byte)
Flow Source IP Address (4 or 16 bytes)
Flow Destination IP Address (4 or 16 bytes)
Flow Feature Segments 1-255

Figure 20. Flow feature packet serialized view, "Flow ID Type" is IPIP

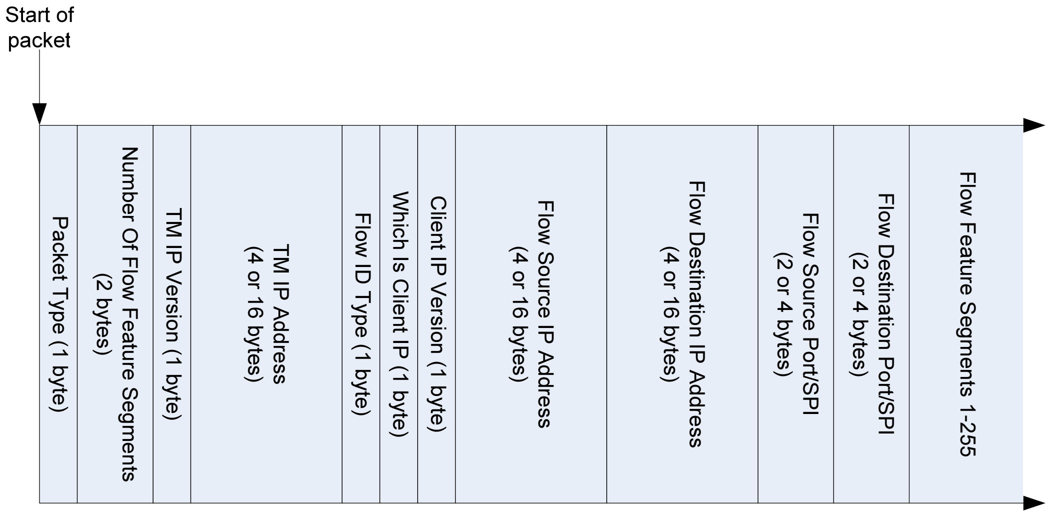
Start of packet
Packet Type (1 byte)
Number Of Flow Feature Segments (2 bytes)
TM IP Version (1 byte)
TM IP Address (4 or 16 bytes)
Flow ID Type (1 byte)
Which Is Client IP (1 byte)
Client IP Version (1 byte)
Flow Source IP Address (4 or 16 bytes)
Flow Destination IP Address (4 or 16 bytes)
Flow Source Port/SPI (2 or 4 bytes)
Flow Destination Port/SPI (2 or 4 bytes)
Flow Feature Segments 1-255

Figure 21. Flow feature packet serialized view, "Flow ID Type" is IPIPPORT or IPIPSPI

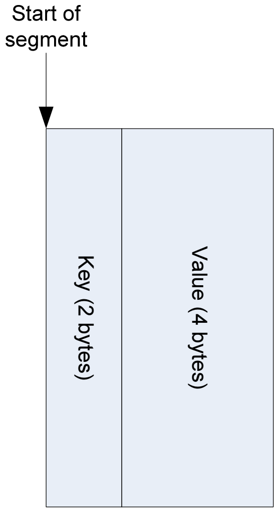

Figure 22. Flow feature segment serialized view

Table 9 summarises each of the FFP fields.

Field Name	Field Length (bytes)	Field Type	Field Description
Packet Type	1	U8	Identifies the packet as an FFP
Number Of Flow Feature Segments	2	U16	Defines how many flow feature segments have been appended to the base FFP. Must be in range [1,255] to be valid
TM IP Version	1	U8	Specifies the IP version of the "TM IP Address" field. A value of 1 corresponds to IPv4 and 2 corresponds to IPv6
TM IP Address	4 or 16	U8 *	Specifies the IP address of the TCAPS manager managing the client this flow belongs to. 4 bytes if "TM IP Version" is 1, 16 bytes if "TM IP Version" is 2
Flow ID Type	1	U8	Specifies the type of flow this FFP is being sent for. A value of 1 corresponds to IPIP, 2 corresponds to IPIPPORT and 3 corresponds to IPIPSPI

Field Name	Field Length (bytes)	Field Type	Field Description
Which Is Client IP	1	U8	Specifies whether the flow source or destination IP address belongs to the TCAPS client being monitored. A value of 1 indicates the flow source IP address belongs to the monitored client. A value of 2 indicates the flow destination IP address belongs to the monitored client
Client IP Version	1	U8	Specifies the IP version of the "Flow Source IP Address" and "Flow Destination IP Address" fields. A value of 1 corresponds to IPv4 and 2 corresponds to IPv6
Flow Source IP Address	4 or 16	U8 *	Specifies the source IP address of the flow this FFP is being sent for. 4 bytes if "Client IP Version" is 1, 16 bytes if "Client IP Version" is 2
Flow Destination IP Address	4 or 16	U8 *	Specifies the destination IP address of the flow this FFP is being sent for. 4 bytes if "Client IP Version" is 1, 16 bytes if "Client IP Version" is 2
Flow Source Port/SPI	2 or 4	U16 or U32	If the "Flow ID Type" field is of type "IPIPPORT" or "IPIPSPI", this field will be present and specifies the source port/SPI of the flow this FFP is being sent for. 2 bytes if "Flow ID Type" is 2, 4 bytes if "Flow ID Type" is 3
Flow Destination Port/SPI	2 or 4	U16 or U32	If the "Flow ID Type" field is of type "IPIPPORT" or "IPIPSPI", this field will be present and specifies the destination port/SPI of the flow this FFP is being sent for. 2 bytes if "Flow ID Type" is 2, 4 bytes if "Flow ID Type" is 3

Table 9. Flow feature packet fields

Table 10 summarises each of the flow feature segment fields.

Field Name	Field Length (bytes)	Field Type	Field Description
Key	2	U16	Contains the binary key for this flow feature segment
Value	4	float	Contains the value associated with the "Key" for this flow feature segment

Table 10. Flow feature segment fields

A FFP is constructed by filling in the appropriate fields in the base packet, and then appending between 1 and 255 flow feature segments to the end of the base packet.

Flow feature segments are structured in a key-value pair arrangement, with a key length of 2 bytes and value length of 4 bytes. They were designed to be used with a binary key, corresponding to the particular flow feature the value field had been calculated for. Using a floating point field type for the flow feature segment value allowed both integer and floating point flow features to be transmitted using this packet type. Discussions with researchers experimenting with classification algorithms revealed that most algorithms use fewer than 100 flow features[2]. Therefore, limiting the key to allow for 65536 different features provides ample room for definition of different flow features that might be used by different algorithms.

---

[2] Informal discussion with Mr. Sebastian Zander of the Centre for Advanced Internet Architectures, Swinburne University

6.1.7 Client Management Packet Type

The client management packet (CMP) type is used to communicate client IP addresses that are to be monitored. It is sent from the TCAPS manager to the TCAPS flow sifter.

The different serialized structures of a CMP are shown in Figure 23, Figure 24 and Figure 25.

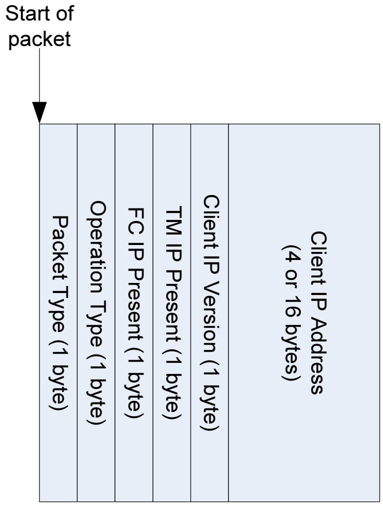

Figure 23. Client management packet serialized view, "FC IP Present" is 0 and "TM IP Present" is 0

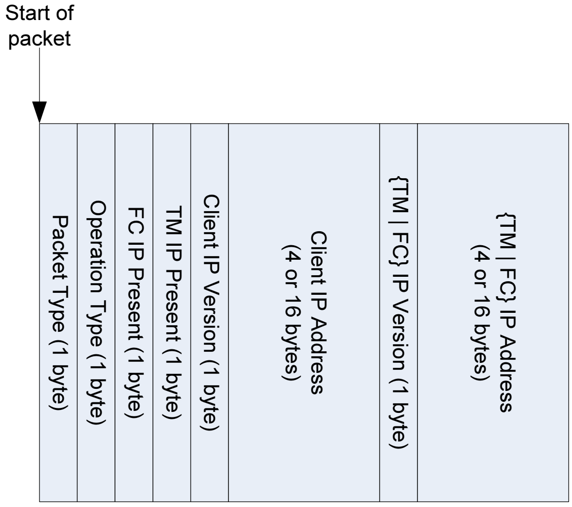
Start of packet
Packet Type (1 byte)
Operation Type (1 byte)
FC IP Present (1 byte)
TM IP Present (1 byte)
Client IP Version (1 byte)
Client IP Address (4 or 16 bytes)
{TM | FC} IP Version (1 byte)
{TM | FC} IP Address (4 or 16 bytes)

Figure 24. Client management packet serialized view, "FC IP Present" is 1 or "TM IP Present" is 1

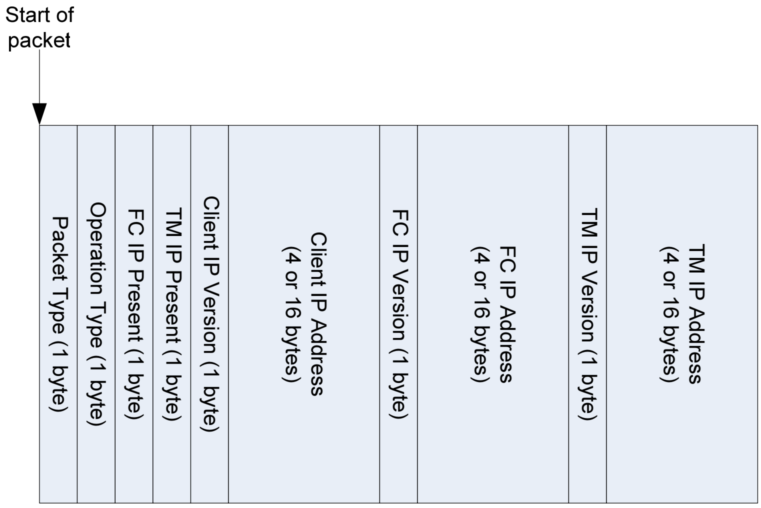

Figure 25. Client management packet serialized view, "FC IP Present" is 1 and "TM IP Present" is 1

Table 11 summarises each of the CMP fields.

Field Name	**Field Length (bytes)**	**Field Type**	**Field Description**
Packet Type	1	U8	Identifies the packet as a CMP
Operation Type	1	U8	Identifies whether the associated client IP is to be added or removed from the current list of monitored clients
FC IP Present	1	U8	Specifies whether this CMP contains the IP address of the TCAPS flow classifier that should be made responsible for classifying this client's flows. A value of 0 corresponds to FALSE and 1 corresponds to TRUE
TM IP Present	1	U8	Specifies whether this CMP contains the IP address of the TCAPS manager that is managing the client. A value of 0 corresponds to FALSE and 1 corresponds to TRUE
Client IP Version	1	U8	Specifies the IP version of the "Client IP Address" field. A value of 1 corresponds to IPv4 and 2 corresponds to IPv6
Client IP Address	4 or 16	U8 *	Specifies the IP address of the client. 4 bytes if "Client IP Version" is 1, 16 bytes if "Client IP Version" is 2
FC IP Version	1	U8	If the "FC IP Present" field is of type "TRUE", this field will be present and specifies the IP version of the "FC IP Address" field. A value of 1 corresponds to IPv4 and 2 corresponds to IPv6
FC IP Address	4 or 16	U8 *	If the "FC IP Present" field is of type "TRUE", this field will be present and specifies the IP address of the TCAPS flow classifier that should be made responsible for classifying this client's flows. 4 bytes if "FC IP Version" is 1, 16 bytes if "FC IP Version" is 2
TM IP Version	1	U8	If the "TM IP Present" field is of type "TRUE", this field will be present and specifies the IP version of the "TM IP Address" field. A value of 1 corresponds to IPv4 and 2 corresponds to IPv6

Field Name	Field Length (bytes)	Field Type	Field Description
TM IP Address	4 or 16	U8 *	If the “TM IP Present” field is of type “TRUE”, this field will be present and specifies the IP address of the TCAPS manager that is managing the client. 4 bytes if “TM IP Version” is 1, 16 bytes if “TM IP Version” is 2

Table 11. Client management packet fields

The 3 different CMP structures relate to the value of the fields “FC IP Present” and “TM IP Present”. When a TCAPS manager uses a CMP to signal a TCAPS flow sifter to monitor a particular client IP address, it has a couple of choices. It can specify the IP address of the TCAPS flow classifier that should be made responsible for classifying flows belonging to the specified client. It can also specify the IP address of the TCAPS manager that is responsible for managing the specified client. Alternatively, it can choose not to specify a TCAPS flow classifier IP address or TCAPS manager IP address, which will let the recipient TCAPS flow sifter decide on appropriate addresses.

Having this flexibility allows a TCAPS manager to tightly control the TCAPS nodes responsible for sifting and classifying specific client flows, if it so desires. Having the ability to specify the TCAPS manager responsible for the client is also advantageous. During the course of normal operation, the TCAPS manager sending the CMP to a TCAPS flow sifter would be responsible for managing the client. However, if a client were to switch TCAPS managers, having the ability to explicitly signal the TCAPS flow sifter and give it a new TCAPS manager IP address may be useful.

6.2 Client Interface

The TCAPS client interface (CI) module is responsible for managing all client side TCAPS related functionality. It communicates with the TCAPS manager using the TCAPS signalling protocol. It is also responsible for communicating with and managing the QoS SS to ensure traffic requiring prioritisation receives it.

The TCAPS client interface module has been developed as an object oriented C++ application, making extensive use of the ACE library. This design choice simplified the prototype development process for the module, at the expense of making it less useful to external CPE manufacturers. CPE devices, which tend to run on embedded hardware platforms, generally have C compilers readily available, but lack other, high-level language development platforms. However, one could speculate that a CPE manufacturer would want to write their own implementation of the CI anyway, on account of wanting to customise it slightly to their particular development environment and hardware platform. With this in mind, the TCAPS prototype CI was written to be somewhat of a reference implementation. Developing the CI in this fashion seemed to be a valid compromise between prototype development simplicity and readiness for use by industry.

The prototype CI demonstrates the core features required by a CI implementation, as well hinting at some additional functionality that could be implemented. A CPE manufacturer could take this implementation, study how it works, and then implement their own from scratch. Alternatively, if they had a C++ development environment for their platform, they could take the reference implementation and extend it as required.

The code structure of the CI is very simple. Figure 26 shows the UML class diagram for the CI. Note that while “Main” is not actually a C++ class within the CI code (“Main” refers to the main() function of the program), it has been included in the diagram for clarity.

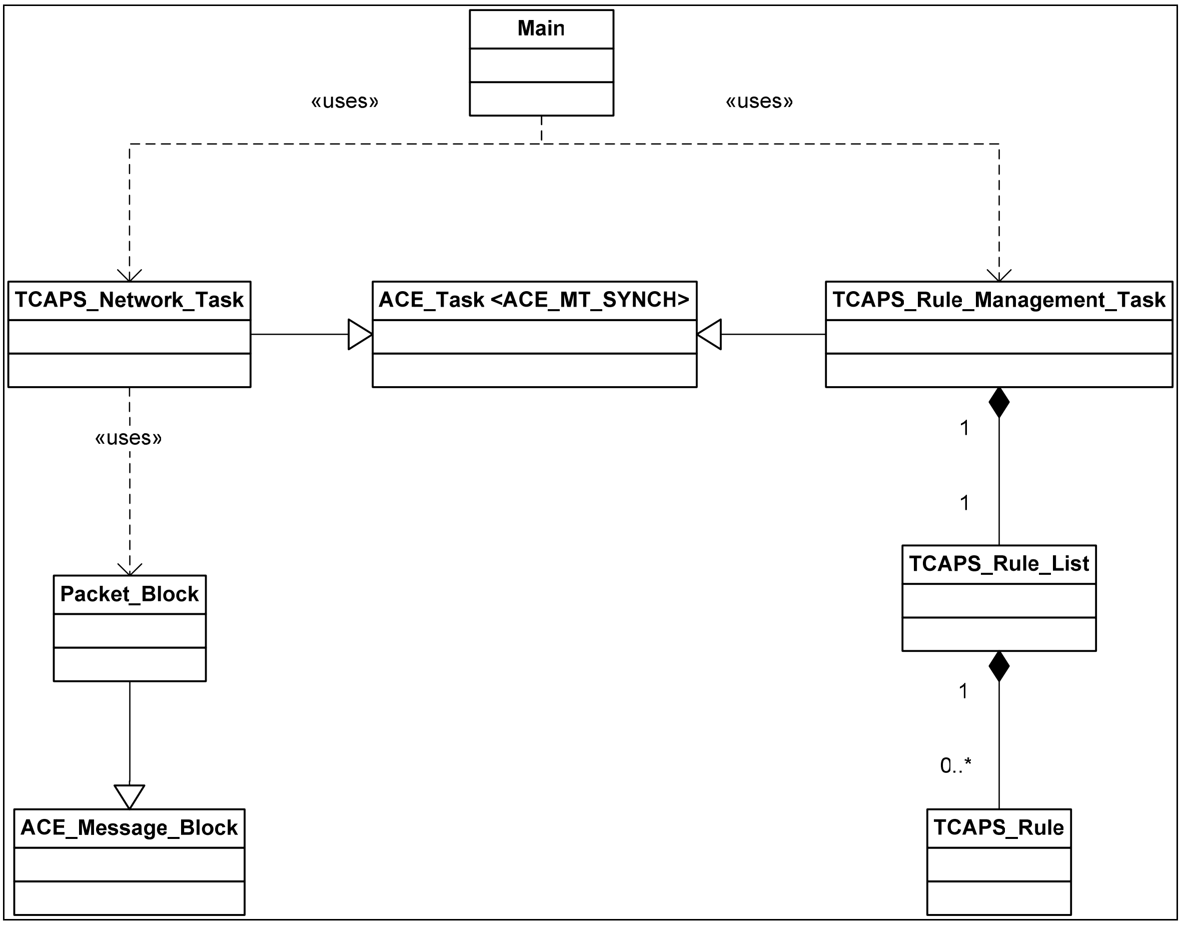

Figure 26. TCAPS client interface UML class diagram

Figure 27 shows a logical representation of the CI threads of execution, and the interaction between these threads and external entities. Arrow heads point in the direction of communication.

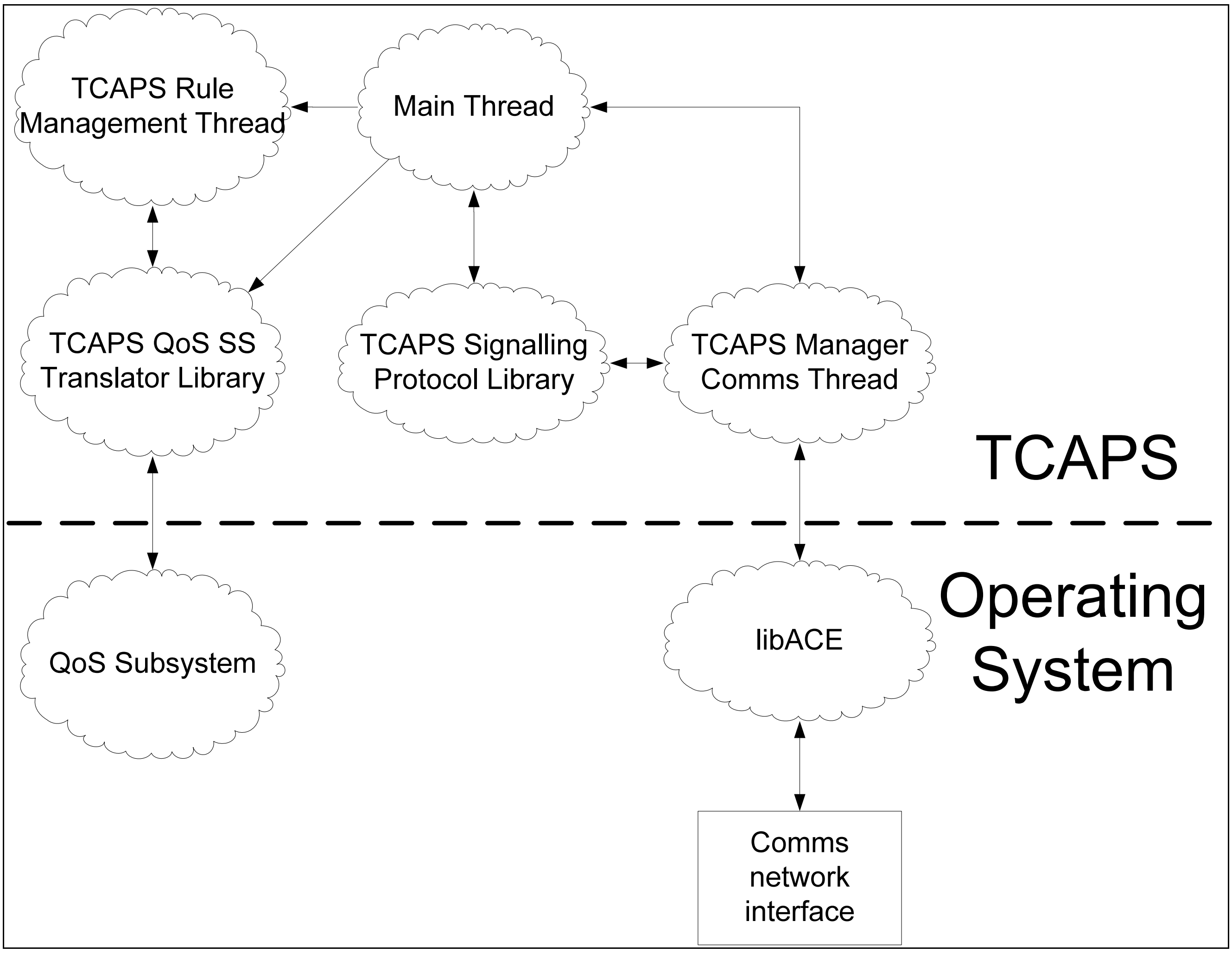
TCAPS Rule Management Thread
Main Thread
TCAPS QoS SS Translator Library
TCAPS Signalling Protocol Library
TCAPS Manager Comms Thread
TCAPS
Operating System
QoS Subsystem
libACE
Comms network interface

Figure 27. TCAPS client interface execution and interaction diagram

The basic operation of the CI can be summarised as follows:

- Program begins executing in the “Main Thread”.
- The “TCAPS Rule Management Thread” and “TCAPS Manager Comms Thread” are created from the “TCAPS_Rule_Management_Task” and “TCAPS_Network_Task” classes respectively.
- The “TCAPS Manager Comms Thread” is responsible for managing the transmission and receipt of all signalling protocol packets from all TCAPS managers the client communicates with.
- The “TCAPS Rule Management Thread” is responsible for managing the list of rules assigned to this client, and liaising with the QoS SS via the “TCAPS QoS SS Translator Library”. In order to add/remove a rule to/from the QoS SS, the “TCAPS Rule Management Thread” uses the “TCAPS QoS SS Translator Library” tcaps_translate_rmp2qosss() function to translate the RMP into a QOS SS specific rule. If it is adding the rule, it then calls tcaps_qosss_add_rule() with the translated command. If it is deleting the rule, it then calls tcaps_qosss_delete_rule() with the translated command.
- The “TCAPS Manager Comms Thread” binds to a randomly allocated UDP port number above 1024.
- The “Main Thread” initialises the QoS SS via the QoS SS translator library.

- The “Main Thread” constructs a signalling protocol poll packet using the “TCAPS Signalling Protocol Library” to register with the TCAPS manager. The IP address of the TCAPS manager is currently hard coded into the module, but will eventually be moved into a runtime configuration file.
- The “Main Thread” passes the poll packet to the “TCAPS Manager Comms Thread”, which then sends the packet to the TCAPS manager.
- At this stage, the CI is registered with the manager, and waits for signalling information from the manager.
- Rule management related signalling information is handed from “TCAPS Manager Comms Thread” to “Main Thread” to “TCAPS_Rule_Management_Task” for processing. Other signalling information is handled by the “Main Thread”.
- The “Main Thread” periodically constructs and sends a poll packet to the TCAPS manager as a keep alive mechanism. The time between updates is currently hard coded into the module, but will eventually be moved into a runtime configuration file.
- On shutdown, the “Main Thread” constructs and sends a poll packet to unregister with the TCAPS manager. It also removes all TCAPS related rules from the QoS SS and shuts the QoS SS down.

The method for disseminating the TCAPS manager IP address to the CI is the most important issue remaining to be solved for the CI. The solution currently thought to be the most suitable involves using DHCP. Most ISPs uses DHCP to dynamically assign IP addresses to their clients when their CPE comes online. Even ISPs that offer static IP addresses to clients use DHCP as the means to notify the CPE of its static IP.

The DHCP options required by a client are requested in a DHCP request packet. DHCP offer packets are sent in reply to DHCP request packets received from clients. Offer packets contain key-value pair options for things like the IP address assigned to the client, the default gateway, subnet mask, DNS server etc.. We could add an additional option to the DHCP request packet, that requested the “tcaps-manager” option from the server. If the ISP was TCAPS enabled, the ISP’s DHCP server would be configured to respond to the “tcaps-server” option by providing a list of IP addresses of active TCAPS managers at the ISP. These IP addresses would then be used to form a list of available TCAPS managers in the TCAPS CI, which would solve the aforementioned problem and give the CI the ability to switch managers if the one currently registered with were to go down.

It is worth noting that if a device does not request an option in its DHCP request packet, the DHCP offer will not contain a key-value pair for that option. This fact ensures that TCAPS unaware CPE will not receive DHCP offer packets with TCAPS related options, as they would not be requesting the new TCAPS options in their DHCP request packets. This would allow TCAPS aware and TCAPS unaware CPE to coexist within the same ISP simultaneously, which satisfies the TCAPS design goals.

There are plenty of other options that could be considered as well. For example, TCAPS aware CPE could be programmed to treat the default gateway as a TCAPS manager and direct all signalling communication to it. The device functioning as the ISP’s default gateway could then perhaps act as a proxy to the real TCAPS managers and forward packets on to the correct machines.

At this stage, a trial of the DHCP solution is on top of the TCAPS “to do” list. Other solutions to this problem will be sought should this option prove to be unfeasible.

6.3 QoS Subsystem Translator

The TCAPS QoS subsystem (QoS SS) translator module provides the interface between the client interface and underlying QoS SS. It allows the client interface to programmatically initialise, shutdown and manage the QoS SS in a QoS SS independent manner. It is also responsible for translating RMPs into QoS SS specific rules.

The TCAPS QoS SS translator module has been implemented as a procedural C library. This design choice ensured that if any external parties were to become involved in the commercialisation of TCAPS, the translator code base would most likely be compatible with their development platforms. This is particularly pertinent to CPE devices, which tend to run on embedded hardware platforms. Such platforms generally have C compilers readily available, but lack other, high-level language development platforms.

The library has been designed to allow a single client interface implementation to be used with multiple QoS SSs. This has been achieved by providing an abstraction layer between generic library function calls and the QoS SS specific function calls. Figure 28 attempts to illustrate these concepts in action, by showing how the most common generic function calls are mapped to QoS SS specific function calls depending on the C pre-processor “#ifdef” commands embedded in the library code.

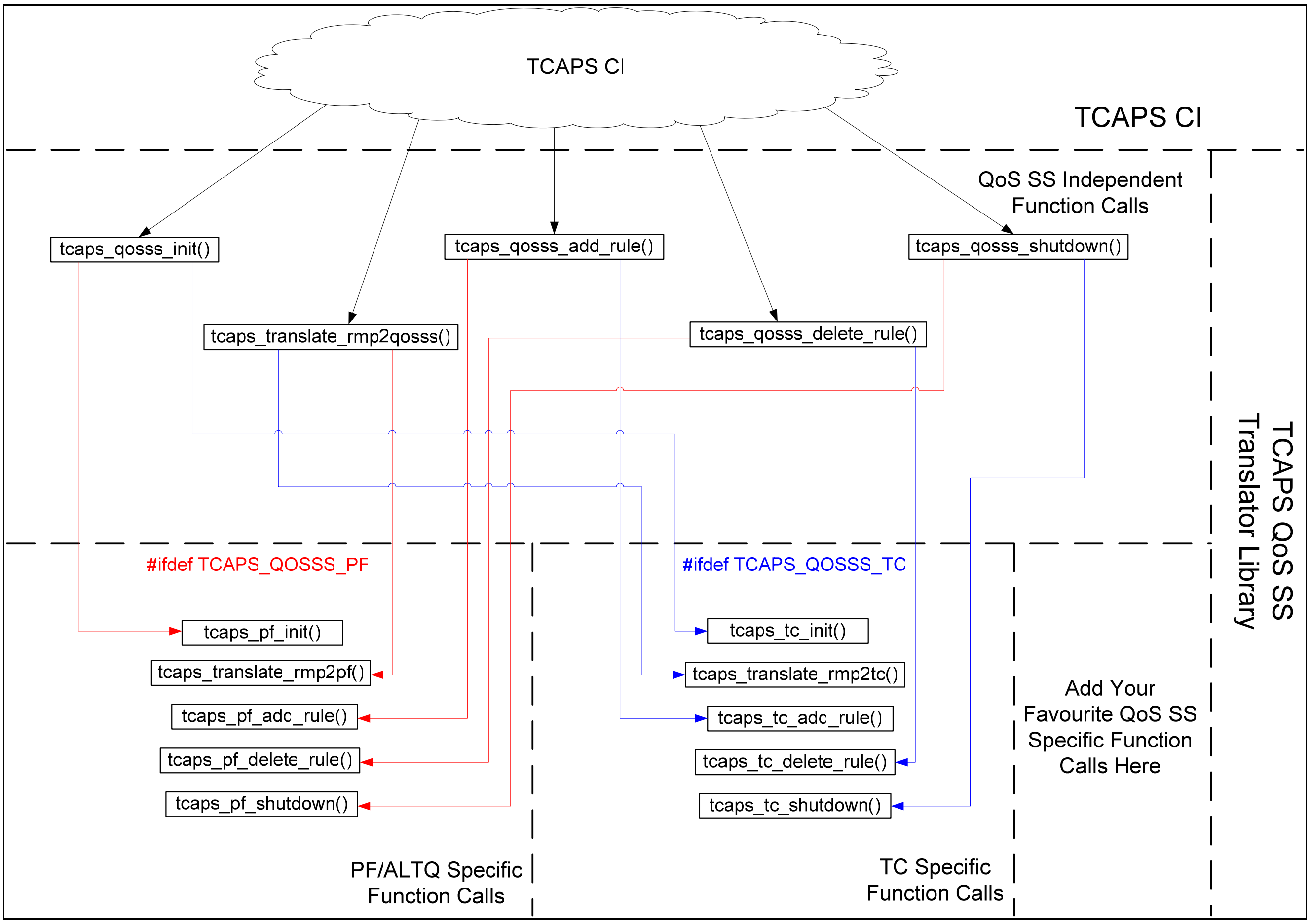

Figure 28. QoS subsystem translator library logical structure and function call diagram

The generic function calls tcaps_qosss_init(), tcaps_translate_rmp2qosss(), tcaps_qosss_add_rule(), tcaps_qosss_delete_rule() and tcaps_qosss_shutdown() are called by code within the TCAPS client interface module. These function calls then call the corresponding QoS SS specific function call to perform the required function. In this way, the TCAPS client interface is not calling any QoS SS specific code directly. This allows us to change the underlying QoS SS without requiring any changes to the TCAPS client interface code at all. This feature would be extremely useful for CPE manufacturers that perhaps had a couple of different CPE models running different QoS SSs. They could write one implementation of the TCAPS client interface to run on all of their equipment models, and only write QoS SS specific changes into the QoS SS translator library. This reduces development time and allows for clean separation between platform specific and platform unspecific differences.

The PF/ALTQ specific function calls have been fully implemented as part of the current TCAPS prototype. The TC [16] specific function calls have not been implemented, but the function stubs have been included in the library for the purposes of understanding how the library can be extended to support new QoS SSs.

The library is designed in such a way that only one QoS SS can be used at any one time. I am unable to think of a situation where using multiple QoS SSs simultaneously would be beneficial, and therefore feel trying to cater for this rare situation would unnecessarily complicate the code. Compiling the library therefore requires the user to define the particular QoS SS to build the library for. By having to do this at compile time, the C pre-processor directives inside the library code modify the library's behaviour to map the generic function calls to the specific function calls for the specified QoS SS. This compilation process has an added bonus that the compiled size of the library is only as large as the compiled size of the generic function calls and particular QoS SS specific function calls. Function calls for other QoS SSs in the library code base are not included in the compiled library, which reduces overall library size. This is not a huge advantage, but is probably useful in embedded device environments where this library will primarily be used, and where code size can be an important issue.

The process for extending the QoS SS translator library to support a new QoS SS is reasonably straight forward. Two new files need to be added to the library source code base. These files should be named "tcaps_<QoS SS name>_translator.c" and "tcaps_<QoS SS name>_translator.h", replacing <QoS SS name> with the name of the QoS SS.

The "tcaps_<QoS SS name>_translator.h" file should, at a minimum, provide a function prototype for all functions available in the "tcaps_qosss_translator.h" file, but replacing "qosss" in their name with "<QoS SS name>".

After this has been done, the following lines need to be added to the "tcaps_qosss_translator.h" file:

```
#ifdef TCAPS_QOSSS_<QoS SS name>
	#include "tcaps_<QoS SS name>_translator.h"
#endif
```

This step ensures the newly defined QoS SS function prototypes are included in the "tcaps_qosss_translator.h" generic header file.

All that remains to be done at this stage is add "#ifdef TCAPS_QOSSS_<QoS SS name> … #endif" code fragments to the generic functions in "tcaps_qosss_translator.c" in order to ensure the mapping of generic function calls to the new QoS SS function calls occurs. The developer can then implement the required QoS SS specific functions within the "tcaps_<QoS SS name>_translator.c" file to perform the required functionality.

6.4 Flow Sifter

The TCAPS flow sifter (FS) module is responsible for inspecting TCAPS client traffic, sifting the traffic into flows, calculating statistical features for each flow and sending these features to a TCAPS flow classifier for classification. It communicates with the TCAPS manager and TCAPS flow classifier using the TCAPS signalling protocol.

The TCAPS flow sifter module has been developed as an object oriented C++ application, making extensive use of the ACE library. This design choice simplified the prototype development process for the module and provides a sound foundation for future development. Unlike the CI, the FS does not particularly need to run on an embedded platform with limited resources and a cut down operating system. Therefore, the choice to use threading and sockets through the ACE library is justified and not likely to cause any problems in the future. The previously discussed benefits of using the ACE library also lend credence to this design choice.

Figure 29 shows the UML class diagram for the FS. Note that while “Main” is not actually a C++ class within the FS code (“Main” refers to the main() function of the program), it has been included in the diagram for clarity.

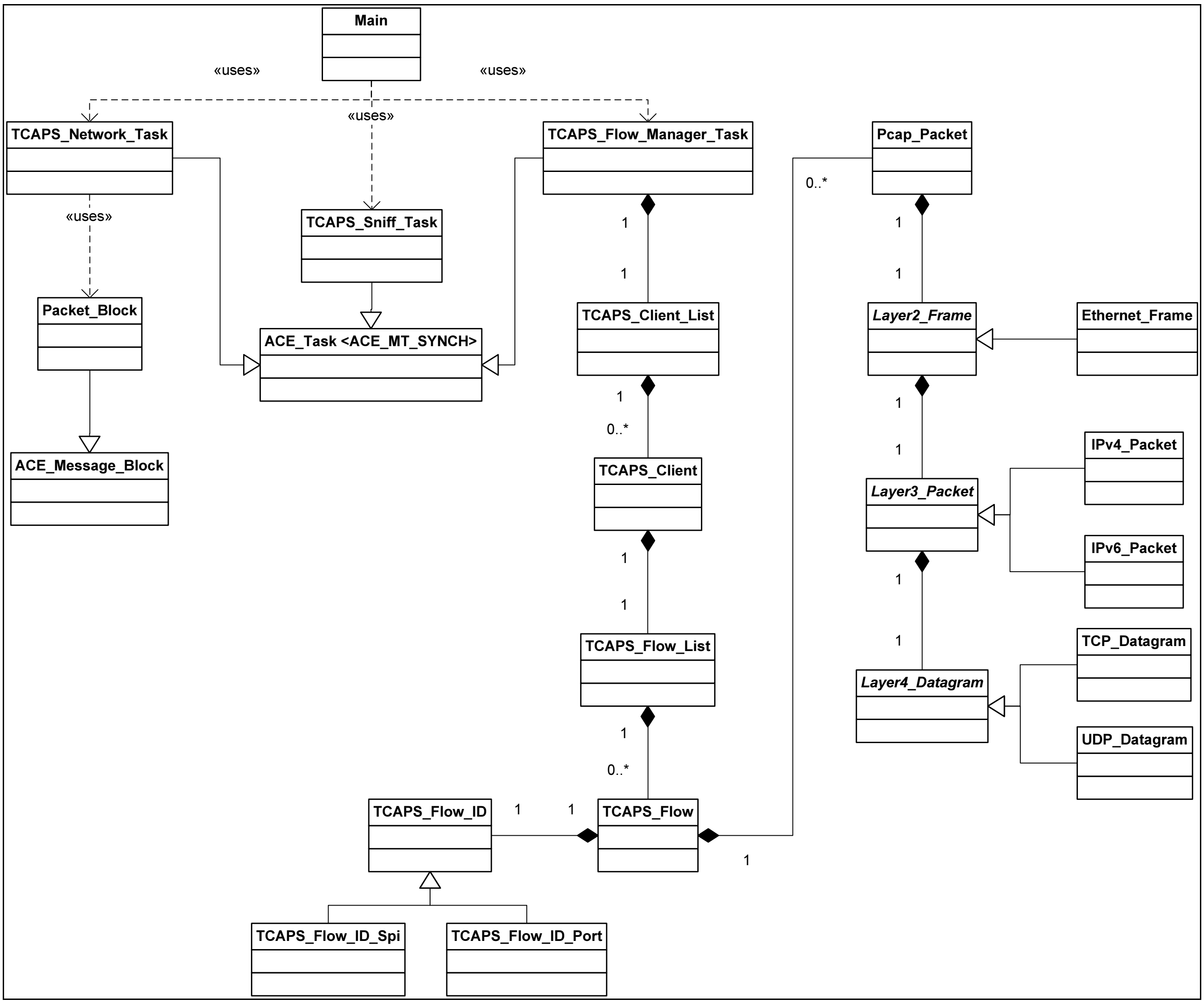

Figure 29. TCAPS flow sifter UML class diagram

Figure 30 shows a logical representation of the FS threads of execution, and the interaction between these threads and external entities. Arrow heads point in the direction of communication.

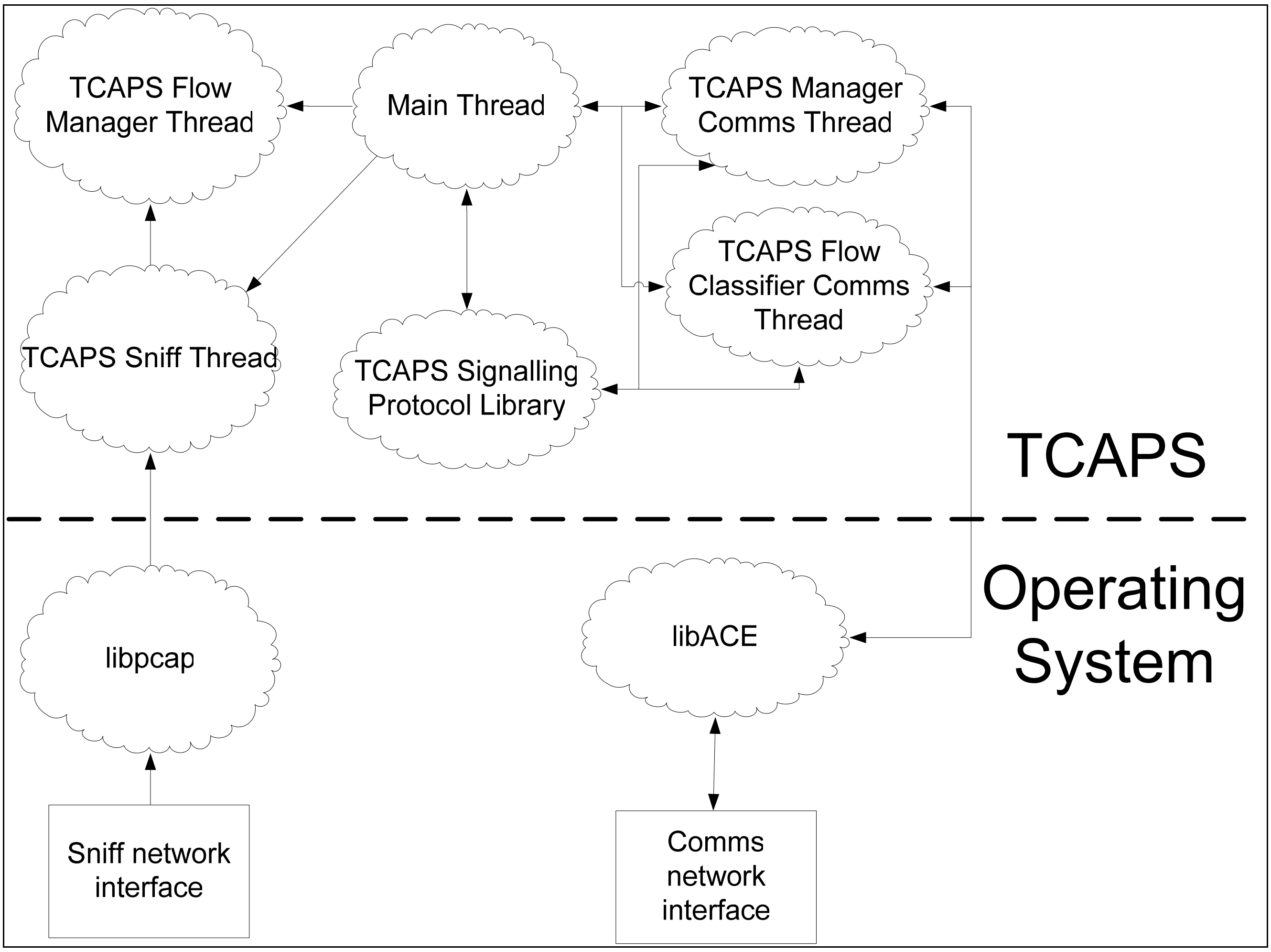
TCAPS Flow Manager Thread
Main Thread
TCAPS Manager Comms Thread
TCAPS Flow Classifier Comms Thread
TCAPS Sniff Thread
TCAPS Signalling Protocol Library
TCAPS
Operating System
libpcap
libACE
Sniff network interface
Comms network interface

Figure 30. TCAPS flow sifter execution and interaction diagram

The basic operation of the FS can be summarised as follows:

- Program begins executing in the “Main Thread”.
- The “Main Thread” initialises the libpcap live sniffing session.
- The “TCAPS Flow Manager Thread”, “TCAPS Sniff Thread”, “TCAPS Manager Comms Thread” and “TCAPS Flow Classifier Comms Thread” are created from the “TCAPS_Flow_Manager_Task”, “TCAPS_Sniff_Task”, “TCAPS_Network_Task” and “TCAPS_Network_Task” classes respectively.
- The “TCAPS Manager Comms Thread” is responsible for managing the transmission and receipt of all signalling protocol packets to/from all TCAPS managers the FS communicates with.
- The “TCAPS Flow Classifier Comms Thread” is responsible for managing the transmission and receipt of all signalling protocol packets to/from all TCAPS flow classifiers the FS communicates with.
- The “TCAPS Sniff Thread” is responsible for sniffing packets from libpcap and passing them to the “TCAPS Flow Manager Thread”.
- The “TCAPS Flow Manager Thread” is responsible for identifying packets that belong to valid TCAPS clients, sorting those packets into individual flows and calculating statistical features for each flow as packets are added.

- The “TCAPS Manager Comms Thread” binds to an arbitrarily defined UDP port number, currently 4567. The UDP port number is currently hard coded into the module, but will eventually be moved into a runtime configuration file.
- The “TCAPS Flow Classifier Comms Thread” binds to a randomly allocated UDP port number above 1024.
- For each new client management packet (CMP) received via the “TCAPS Manager Comms Thread”, the “Main Thread” will add an entry to the “TCAPS_Client_List” list in the “TCAPS Flow Manager Thread”. Each client monitored by the FS is assigned to a TCAPS flow classifier that will receive all flow feature packets relating to the client’s flows.
- The “Main Thread” periodically constructs flow feature packets for each client flow that has changed since the last time this operation was performed. The flow feature packets are sent to the TCAPS flow classifier that the client was assigned to. The time between updates is currently hard coded into the module, but will eventually be moved into a runtime configuration file.
- If an existing client flow does not transmit any traffic within a given threshold time, the client flow will be removed by the FS and all associated resources released. The threshold time is currently hard coded into the module, but will eventually be moved into a runtime configuration file.
- On shutdown, the “TCAPS Sniff Thread” shuts the libpcap session down and “Main Thread” frees all system resources.

One of the more interesting aspects of the FS module worth discussing is the way it calculates features for each client flow currently being monitored. Two broad classes of flow feature have been defined within the FS: “since beginning” and “sliding window”. Flow features in the “since beginning” class are calculated based on every packet that has been observed since the flow began. They are updated each time a new packet arrives for the flow. This appears to be the more traditional way of calculating features, based on discussions with machine learning algorithm researchers[3]. Flow features in the “sliding window” class are calculated based on the last X packets observed for the flow, where X is user configurable. They are updated at periodic intervals rather than each time a packet is observed, and provide more finely grained statistics for a flow.

Features calculated based on a “sliding window” premise may provide better flow classification criteria than “since beginning” features. For example, if an application were to have periods where its communications change from being non-realtime to realtime, “since beginning” flow features would probably miss the change in traffic characteristics. However, “sliding window” features would most likely pick up the change, assuming the number of packets in the window is not too large. Therefore, the view was taken that it would perhaps be beneficial to make the effort to implement the “sliding window” features, in the hope that some experimentation may be performed at a later date to examine their characteristics and usefulness in flow classification.

6.5 Flow Classifier

The TCAPS flow classifier (FC) module is responsible for classifying realtime/interactive client flows and notifying the TCAPS manager responsible for the client about the identified realtime/interactive flows so that the client can be instructed to prioritise the flow. The flow classifier module communicates with the TCAPS flow sifter and TCAPS manager using the TCAPS signalling protocol.

The TCAPS flow classifier module has been developed as an object oriented C++ application, making extensive use of the ACE library. This design choice simplified the prototype development process for

---

[3] Informal discussion with Mr. Sebastian Zander of the Centre for Advanced Internet Architectures, Swinburne University

the module and provides a sound foundation for future development. Unlike the CI, the FC does not particularly need to run on an embedded platform with limited resources and a cut down operating system. Therefore, the choice to use threading and sockets through the ACE library is justified and not likely to cause any problems in the future. The previously discussed benefits of using the ACE library also lend credence to this design choice.

Figure 31 shows the UML class diagram for the FC. Note that while “Main” is not actually a C++ class within the FC code (“Main” refers to the main() function of the program), it has been included in the diagram for clarity.

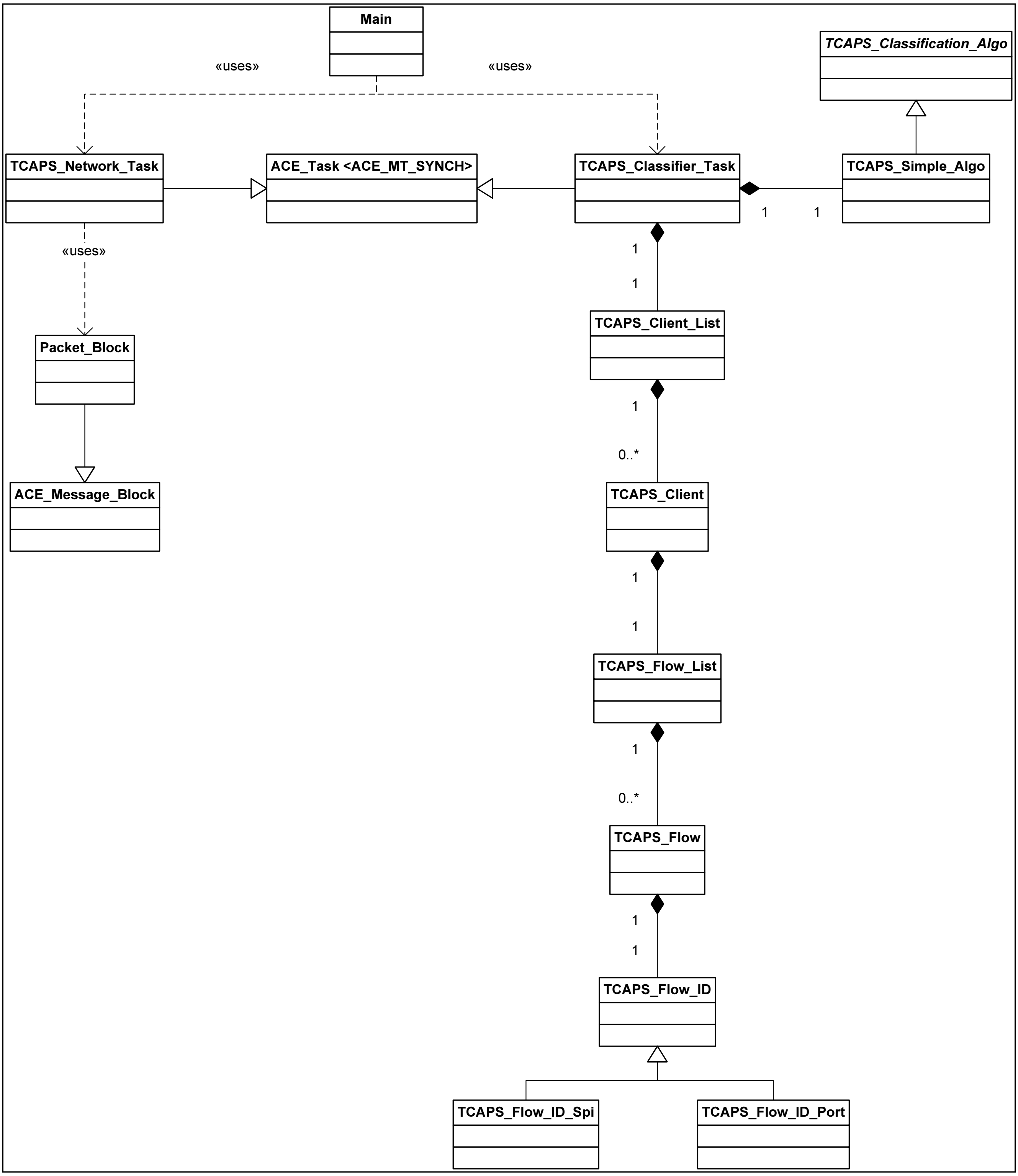

Figure 31. TCAPS flow classifier UML class diagram

Figure 32 shows a logical representation of the FC threads of execution, and the interaction between these threads and external entities. Arrow heads point in the direction of communication.

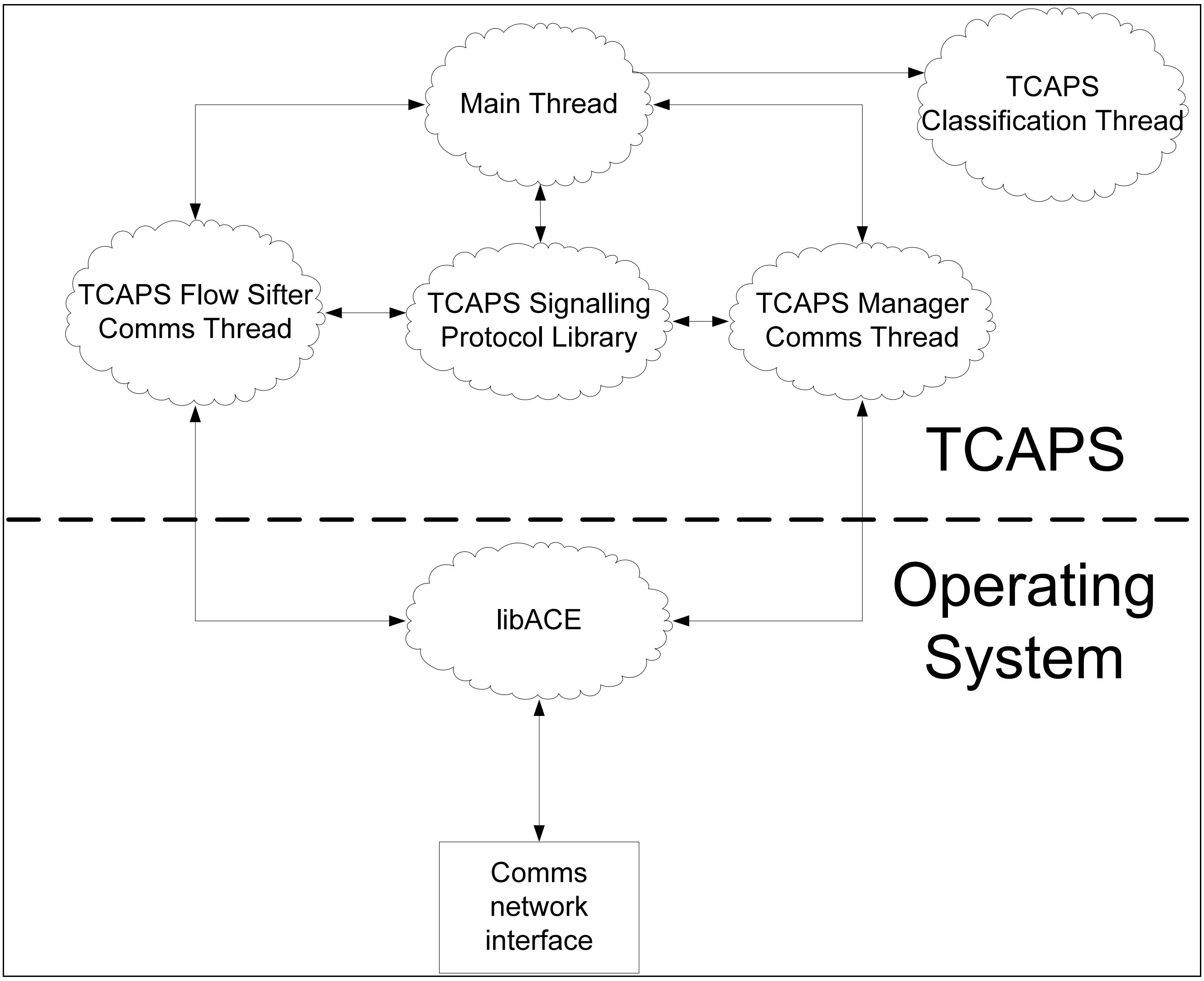
Main Thread
TCAPS Classification Thread
TCAPS Flow Sifter Comms Thread
TCAPS Signalling Protocol Library
TCAPS Manager Comms Thread
TCAPS
Operating System
libACE
Comms network interface

Figure 32. TCAPS flow classifier execution and interaction diagram

The basic operation of the FC can be summarised as follows:

- Program begins executing in the "Main Thread".
- The "TCAPS Classification Thread", "TCAPS Manager Comms Thread" and "TCAPS Flow Sifter Comms Thread" are created from the "TCAPS_Classifier_Task", "TCAPS_Network_Task" and "TCAPS_Network_Task" classes respectively.
- The "TCAPS Manager Comms Thread" is responsible for managing the transmission and receipt of all signalling protocol packets to/from all TCAPS managers the FC communicates with.
- The "TCAPS Flow Sifter Comms Thread" is responsible for managing the transmission and receipt of all signalling protocol packets to/from all TCAPS flow sifters the FC communicates with.
- The "TCAPS Classification Thread" is responsible for classifying realtime/interactive flows and assigning them a priority.
- The "TCAPS Flow Sifter Comms Thread" binds to an arbitrarily defined UDP port number, currently 7777. The UDP port number is currently hard coded into the module, but will eventually be moved into a runtime configuration file.
- The "TCAPS Manager Comms Thread" binds to a randomly allocated UDP port number above 1024.

- For each new flow feature packet (FFP) received via the “TCAPS Flow Sifter Comms Thread”, the “Main Thread” will add an entry to the “TCAPS_Client_List” list in the “TCAPS Classification Thread” if the client the flow belongs to does not exist. If the client already exists but the FFP is for a new flow, the “Main Thread” will add an entry to the client’s “TCAPS_Flow_List”. If the client and flow already exist, the “Main Thread” will update the flow features for the flow to the values contained in the FFP. The IP address of the TCAPS manager managing the client is extracted from the FFP and stored for subsequent communications between the FC and the client’s manager.
- The “Main Thread” periodically checks for any client flows that have had their assigned priority changed since the last check. Flows with a changed priority trigger a new classifier rule management packet (CRMP) to be constructed and sent via the “TCAPS Manager Comms Thread” to the TCAPS manager responsible for managing the client.
- If an existing client does not receive a FFP (either for a new or existing flow) within a given threshold time, the client will be removed by the FC and all associated resources released. The threshold time is currently hard coded into the module, but will eventually be moved into a runtime configuration file.
- On shutdown, the “Main Thread” frees all system resources.

The flow classifier’s classification subsystem deserves some brief discussion. It is the core of the module and in some regards, can be thought of as the core of TCAPS, as it is ultimately responsible for the classification of realtime/interactive flows without human intervention.

The “TCAPS_Simple_Algo” class was implemented as a temporary solution to demonstrate the detection of realtime/interactive traffic flows. Specifically, the “TCAPS_Simple_Algo” was tuned solely to detect Quake 3 and Cisco 7960G VoIP phone traffic flows. This is obviously not a suitable solution for classifying the broad range of realtime/interactive traffic sources in existence. It was always the intention for this module to utilise a machine learning based traffic classifier, which was capable of classifying a broad range of realtime/interactive traffic. However, it was clear early on in the implementation of the FC that such a classifier algorithm would take months to develop on its own, and so the decision was made to specialise the algorithm used in the prototype.

As a result of this, the architecture of the classification subsystem was designed to allow new classification algorithms to be added without requiring any changes to the structure of the FC’s code. This would allow a proper machine learning based algorithm to be developed and added to the module at a later date. This was achieved through the use of C++’s object oriented inheritance capabilities. The only constraint imposed on a new classification algorithm is that it must extend the abstract base class “TCAPS_Classification_Algo”. This ensures that the algorithm must supply a function call with the following prototype:

```
void classify_flow(TCAPS_Flow * flow)
```

The FC code calls this function for each flow requiring classification, and as a result, expects it to be present in any classification algorithm being used in the module.

This ability to easily extend the FC could, in the future, result in a range of classification algorithms being available for selection, which may provide benefits not currently available.

6.6 Manager

The TCAPS manager module is responsible for managing the TCAPS related functionality of TCAPS enabled CPE and communicating with the other ISP side TCAPS components to coordinate required behaviour and operations. It is the only ISP side TCAPS module that communicates with client CPE. It communicates with the TCAPS client interface, TCAPS flow sifter and TCAPS flow classifier using the TCAPS signalling protocol.

The TCAPS manager module has been developed as an object oriented C++ application, making extensive use of the ACE library. This design choice simplified the prototype development process for the module and provides a sound foundation for future development. Unlike the CI, the manager does not particularly need to run on an embedded platform with limited resources and a cut down operating system. Therefore, the choice to use threading and sockets through the ACE library is justified and not likely to cause any problems in the future. The previously discussed benefits of using the ACE library also lend credence to this design choice.

Figure 33 shows the UML class diagram for the manager. Note that while “Main” is not actually a C++ class within the manager code (“Main” refers to the main() function of the program), it has been included in the diagram for clarity.

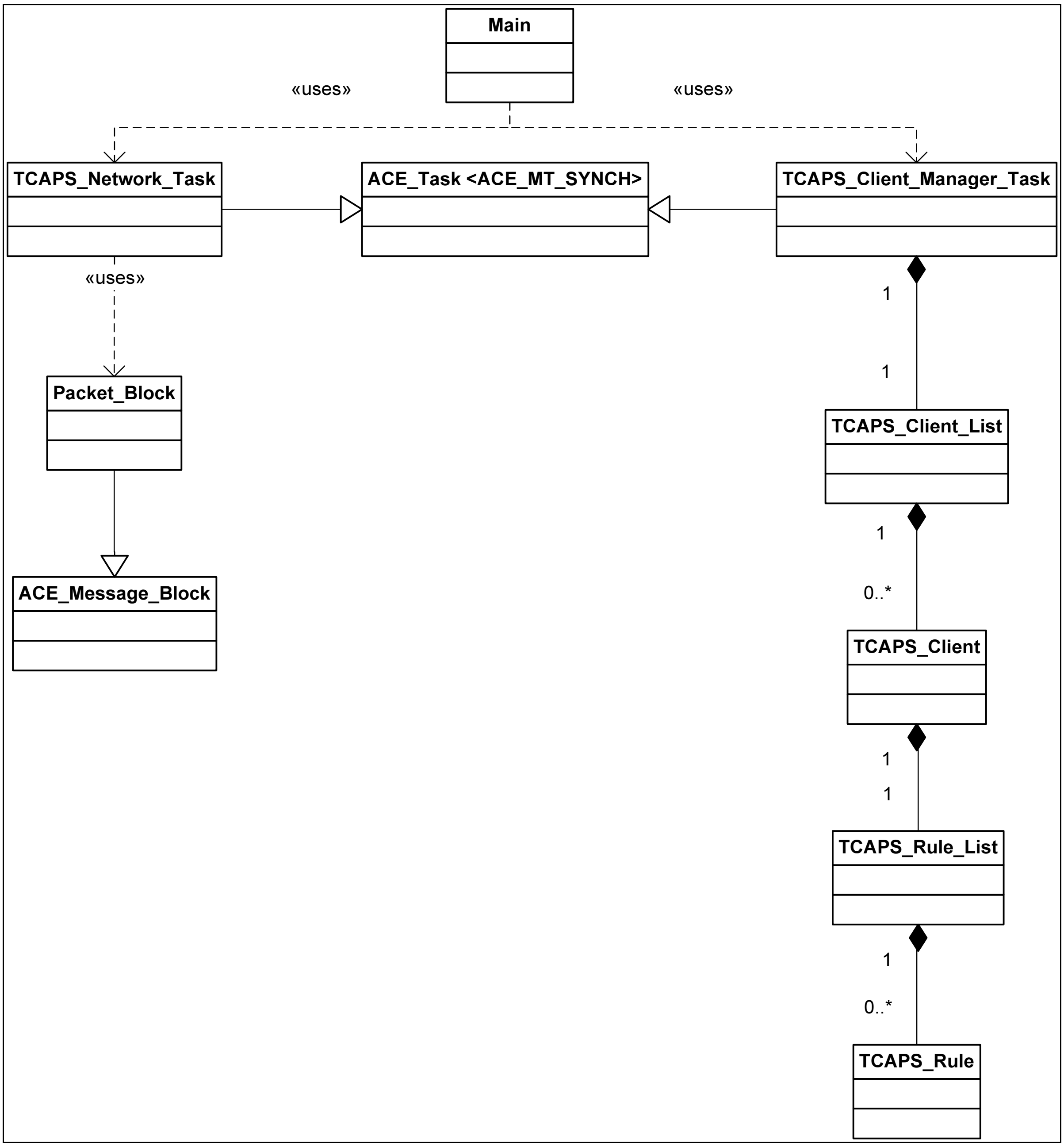

Figure 33. TCAPS manager UML class diagram

Figure 34 shows a logical representation of the manager threads of execution, and the interaction between these threads and external entities. Arrow heads point in the direction of communication.

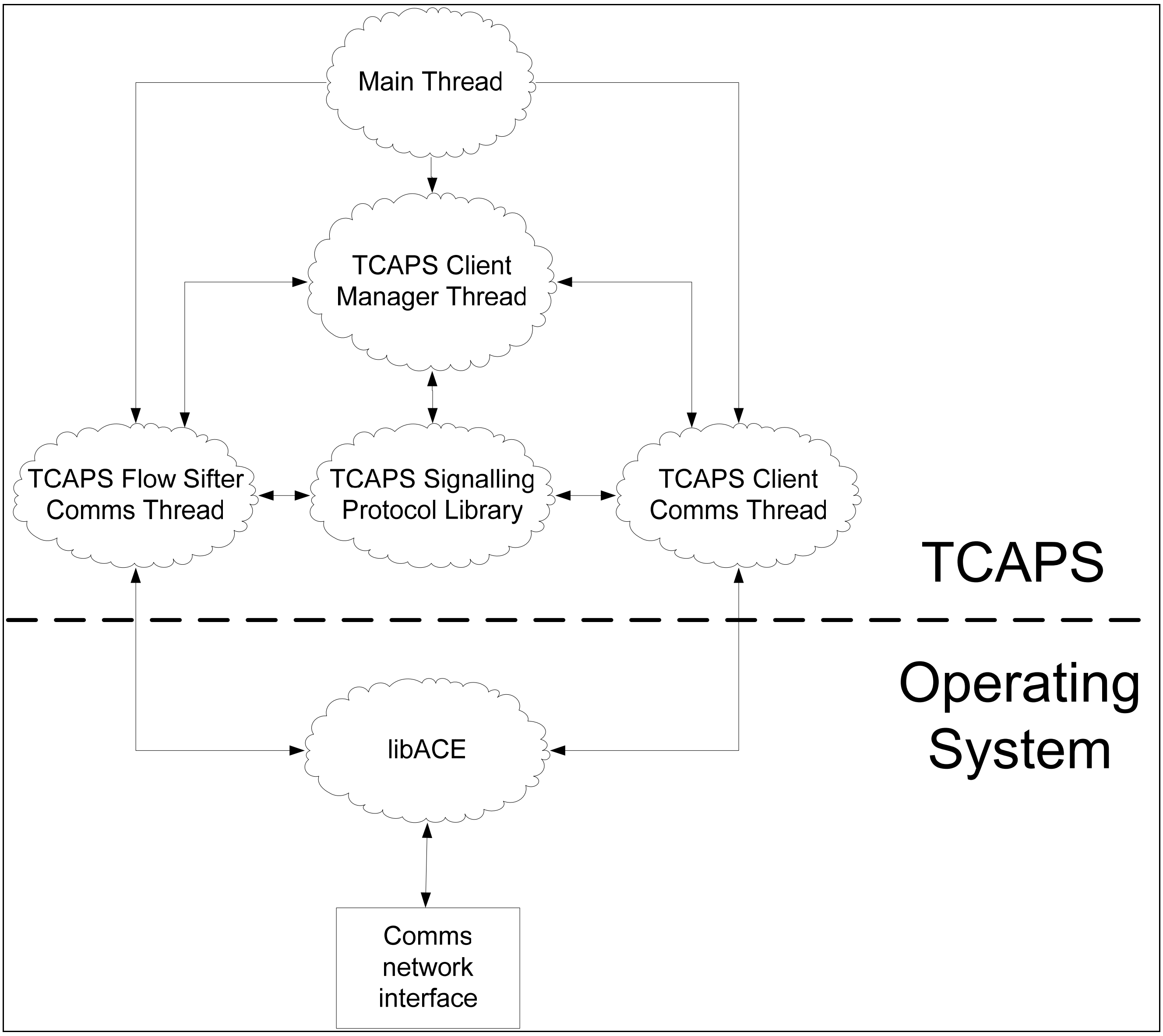
Main Thread
TCAPS Client Manager Thread
TCAPS Flow Sifter Comms Thread
TCAPS Signalling Protocol Library
TCAPS Client Comms Thread
TCAPS
Operating System
libACE
Comms network interface

Figure 34. TCAPS manager execution and interaction diagram

The basic operation of the manager can be summarised as follows:

- Program begins executing in the “Main Thread”.
- The “TCAPS Client Manager Thread”, “TCAPS Flow Sifter Comms Thread” and “TCAPS Client Comms Thread” are created from the “TCAPS_Client_Manager_Task”, “TCAPS_Network_Task” and “TCAPS_Network_Task” classes respectively.
- The “TCAPS Flow Sifter Comms Thread” is responsible for managing the transmission and receipt of all signalling protocol packets to/from all TCAPS flow sifters the manager communicates with.
- The “TCAPS Client Comms Thread” is responsible for managing the transmission and receipt of all signalling protocol packets to/from all TCAPS enabled CPE the manager communicates with.
- The “TCAPS Client Manager Thread” is responsible for handling client management tasks such as registration, deregistration and rule management.

- The “TCAPS Client Comms Thread” binds to an arbitrarily defined UDP port number, currently 6789. The UDP port number is currently hard coded into the module, but will eventually be moved into a runtime configuration file.
- The “TCAPS Flow Sifter Comms Thread” binds to a randomly allocated UDP port number above 1024.
- For each new registration poll packet (PP) received via the “TCAPS Client Comms Thread”, the “Main Thread” will add an entry to the “TCAPS_Client_List” list in the “TCAPS Client Manager Thread”. The client is then assigned to a TCAPS flow sifter and a CMP is sent to the assigned flow sifter.
- If a flow belonging to one of the existing clients is classified as realtime/interactive, the manager receives a CRMP from the flow classifier. The encapsulated RMP is extracted from the CRMP, the empty fields are filled in by the manager and the RMP is then sent to the appropriate client. A copy of the RMP is also stored by the manager to facilitate rule transfer requests.
- If an existing client does not send a PP to the manager within a given threshold time, the client will be removed by the manager and all associated resources released. The threshold time is currently hard coded into the module, but will eventually be moved into a runtime configuration file.
- On shutdown, the “Main Thread” frees all system resources.

7. Testing & Outcomes

Unfortunately, time constraints limited the amount of testing that could be performed on the TCAPS system prototype. The original test plan involved using the Broadband Access Research Testbed (BART) at the Centre for Advanced Internet Architectures (CAIA) to fully evaluate the prototype over a typical broadband access network. As a result of the time constraints, the original test plan was discarded and a smaller testbed was built to allow rapid test set up and execution. Figure 35 shows the testbed used to test the TCAPS system prototype.

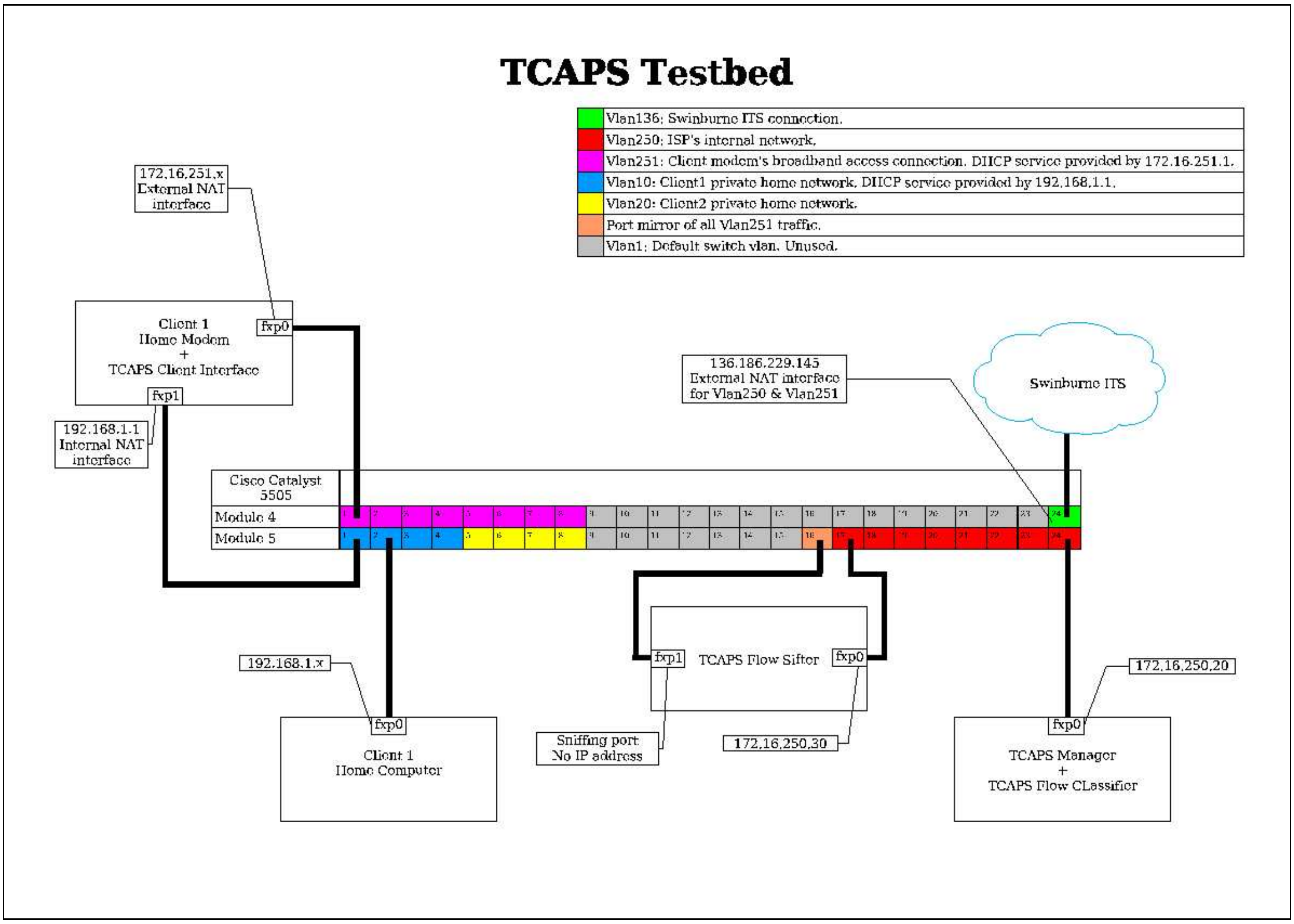
TCAPS Testbed
Vlan136: Swinburne ITS connection.
Vlan250: ISP's internal network.
Vlan251: Client modem's broadband access connection. DHCP service provided by 172.16.251.1.
Vlan10: Client1 private home network. DHCP service provided by 192.168.1.1.
Vlan20: Client2 private home network.
Port mirror of all Vlan251 traffic.
Vlan1: Default switch vlan. Unused.
172.16.251.x External NAT interface
Client 1 Home Modem + TCAPS Client Interface
fxp0
fxp1
192.168.1.1 Internal NAT interface
136.186.229.145 External NAT interface for Vlan250 & Vlan251
Swinburne ITS
Cisco Catalyst 5505
Module 4
Module 5
192.168.1.x
fxp0
Client 1 Home Computer
fxp1
TCAPS Flow Sifter
fxp0
Sniffing port No IP address
172.16.250.30
172.16.250.20
fxp0
TCAPS Manager + TCAPS Flow CLassifier

Figure 35. TCAPS testbed

A Cisco Catalyst 5505 switch running a supervisor II module and route switch module was used to emulate the client LAN, broadband access network, ISP network and connection to the wider Internet. VLANs provided logical groupings of 10/100Mbit Ethernet ports that were used to provide the various connections required. The blue ports represented a home LAN switch, connecting a home computer with the broadband access CPE. The broadband access modem was emulated using a standard PC running FreeBSD. It ran the PF/ALTQ QoS subsystem and the TCAPS client interface modules to behave as a standard CPE would.

The home modem used NAT to allow the devices on the 192.168.1.x "home network" (blue ports) to connect to the ISP via the pink port connection. The pink ports emulate connection points into the broadband access network, and devices that connect to these are assigned IP addresses in the 172.16.251.x range by DHCP.

The orange port provided a network tap of all traffic passing through the pink ports. This was used by the TCAPS flow sifter machine to observe all client traffic.

The red ports emulate the ISPs internal network, with IP address range 172.16.250.x assigned by DHCP. The decision to run the flow sifter on its own physical machine was made as a result of the significantly higher hardware requirements (memory and processing) required by the software, compare to the manager and flow classifier.

The green port emulates the ISPs internet connection, which in this case was a connection to Swinburne's corporate network. As there were multiple devices running on the testbed using private IP addresses, the connection to Swinburne's network had to use NAT in order to provide "Internet" connectivity to all testbed devices.

Finally, the Catalyst 5505 itself performed the role of ISP router, by routing packets between the client side (pink ports) and ISP side (red ports) networks.

This testbed effectively emulates the network shown in Figure 3 page 12, except the flow classifier and manager have been split from the flow sifter in the testbed.

The tests performed using the testbed were primarily aimed at testing for functional correctness, as this was the most critical requirement for producing a demonstrable prototype. The functional testing process was performed iteratively, and uncovered numerous bugs within the system implementation. Considerable effort was made to correct each uncovered bug, which reduced the time available to performing other testing aimed at identifying the operational limits of the prototype.

Two operational limits tests were performed to answer the questions:

- Could the system prototype handle 100Mbit line rate traffic running on commodity hardware?
- Could the system prototype classify Quake 3 and/or Cisco 7960G VoIP phone traffic flows within the required 20 second time frame?

Due to the time constraints, the results for these tests were not quantitatively measured, but rather empirically observed. Based on the observations, the answer to both of the test questions was found to be "Yes". Flow sifting and classification was able to function at 100Mbit traffic rates on commodity hardware. Flow classification of Quake 3 flows and Cisco 7960G VoIP phone traffic flows was typically occurring within 5 seconds of the flow beginning.

It should be noted that these tests were performed with relatively low numbers of concurrent flows (10 or fewer). It is certain that running these tests with greater numbers of concurrent flows would adversely affect the performance of the prototype system, based on the internal software architecture of the TCAPS components. However, the severity of the effects cannot be easily estimated, and only further testing would provide an answer as to how many concurrent flows can be supported at 100Mbit line rates.

Whilst these tests verified the prototype met its basic design goals and functioned according to specification, additional testing is obviously required to provide a more comprehensive analysis of the prototype's operational limits and areas for improvement.

8. Current TCAPS Limitations

The design and development process for TCAPS resulted in a huge number of decisions being made about a wide range of implementation issues. Decisions tend to result in the lesser of two evils being chosen, at the expense of the positive aspects of the other choice. Some of the decisions made for the implementation of the TCAPS prototype have resulted in system limitations that need to be documented as caveats for its use.

The decision to implement the client interface in C++ like the other TCAPS modules limits the possibility of broadband equipment manufacturers being able to use the code base. As previously discussed, the platform most CPE devices run on is embedded, which tend not to have higher level language development platforms available for them. The CI's reliance on libACE and socket communications further restricts the deployment options for the CI.

In order to have a have a functional prototype ready in time, the issue of TCAPS module IP address dissemination was not tackled. Currently, the IP address of the TCAPS manager is hard coded into the CI and the IP address of the FC is hard coded into the FS. The proposed solution to this problem discussed previously involves the use of new DHCP options requested/sent as part of the DHCP protocol exchange.

The minimal security implemented as part of TCAPS is currently insufficient. TCAPS currently requires an external security framework such as IPsec to ensure the security of signalling protocol transmissions. Other options like HTTPS at the CI end could also provide a possible security framework to integrate into TCAPS. The decision to use IP based signalling ensured that an external security framework would be more likely to fit right in to the current architecture rather than having to significantly change things around. It is proposed at this time that the client's ISP authentication details (RADIUS username and password perhaps) are used as the basis for authentication to TCAPS, although this integration has not yet been planned or designed in any way.

Priority queuing, whilst a good first step, does not allow finely grained QoS. We saw in section 5.2.1 that whilst priority queuing is able to constrain the maximum queuing delay experience by a prioritised packet, it is not able to guarantee a fixed queuing delay less than the maximum. If we could reduce the size of the largest packet placed in the non-priority, we would be able to guarantee a reduced maximum queuing delay, proportional to the serialization delay of the largest packet size. Ubicom's StreamEngine technology uses IP fragmentation to achieve this reduction of non-priority packet size to improve its priority queuing maximum delay. The TCAPS QoS SS would benefit greatly if it could perform similar functionality to StreamEngine's IP fragmentation. However, reinventing the wheel is never the right choice if it can be avoided. Therefore integrating the TCAPS CI into a StreamEngine enabled CPE device would provide a faster solution to this current TCAPS limitation as opposed to developing our own, similar technology.

The classification algorithm currently implemented in the flow classifier was never meant to be a solution that could evolve into a commercially ready piece of software. It was implemented to specifically target Quake 3 and Cisco 7960G VoIP phone traffic streams, as opposed to realtime games and VoIP in general. This specialisation allowed the prototype to demonstrate its capabilities without requiring a significant time investment in creating a generic flow classifier capable of detecting any realtime/interactive traffic. The plan for the flow classifier was always to implement a machine learning based classification algorithm after the prototype had proved the TCAPS concept was feasible. The design of the flow classifier enables a new classification algorithm to be slotted into the place where the “Simple Algorithm” currently resides in the code, without any structural change to the remainder of the flow classifier code base.

The flow sifter’s packet sniffing functionality currently assumes it is running in an Ethernet OSI layer 2 environment. If the underlying link layer technology of the tap supplied to the flow sifter was ATM for example, the sifter would not be able to function.

The ISP side TCAPS modules currently do not check node status in a similar fashion to the CI polling the manager. The ISP modules should keep a table of active ISP TCAPS nodes and periodically verify the nodes are still functioning, removing them from the active table if they appear to have failed.

9. Further Work

The TCAPS prototype implementation documented in this report is certainly not ready for commercialisation yet. There are a number of hurdles that need to be passed before it would be ready to go to market. There are also an almost infinite number of additional features that could be added, a few of which will be discussed below. None of the hurdles are significant enough that they could not be solved with some additional development work and time.

The current TCAPS code base needs to be tidied up, refined, optimised, and documented before further development work is undertaken. Lack of time and speed of development made proper and accurate documenting of code almost impossible, and resulted in many sections of code being left in sub optimal configurations.

Run time configuration files need to be implemented to allow user configuration of variables that are currently hard coded into the various modules.

Selecting and implementing a means for TCAPS node IP dissemination will an important step towards a commercially ready TCAPS implementation.

Integrating a security framework into or around TCAPS is a critical hurdle that must be achieved.

Implementing an accurate and efficient machine learning based classifier into the TCAPS flow classifier is another important hurdle that must be overcome. If a machine learning classifier looks like it will take too long to perfect, an interim TCP/UDP port inspection based classifier could be implemented as an immediate solution that could be upgraded to a machine learning based alternative at a later date. This option could be used to reduce time to market if required.

Providing a user interface to the system that would allow clients to specify permanent rules would be a useful feature addition to TCAPS.

Adding the ability for TCAPS to exploit more of the features provided by the CPE’s QoS subsystem would be a useful feature addition. For example, as well as providing prioritisation information, TCAPS could also provide firewall rules that are dynamically created and installed at client CPE devices to reduce the impact of an Internet virus.

A non critical piece of further work would be to explore alternatives to priority queuing as means for providing prioritisation.

Getting CPE manufacturers on board and developing their own TCAPS enabled CPE is a crucial step to seeing TCAPS commercialised.

Running an ISP based TCAPS trial would be required once the code had matured to a stable point and was ready for a field trial. If a TCAPS enabled CPE device was not ready for field trials, a standalone UNIX router performing the functionality of a TCAPS enabled CPE could be installed at participating customer premises instead.

10. Conclusion & Recommendations

The Traffic Classification and Prioritisation System (TCAPS) project aimed to develop an architecture for realtime network traffic classification and prioritisation for use in ISP broadband access networks, as well as a prototype of the system. The specification, design and development of the system has taken 1 year to arrive at its current state.

The TCAPS architecture has been designed to provide a robust, scalable and efficient framework for broadband QoS provision. The features offered by the architecture include: deployment flexibility, interoperability between TCAPS enabled and standard CPE, backwards compatibility of TCAPS enabled CPE, automated definition of prioritisation rules, customised prioritisation rules for each client, and the ability to classify encrypted flows.

The current TCAPS prototype built to demonstrate the TCAPS architecture provides a solid foundation for any future development of TCAPS. It has been implemented entirely in software for speed of development and deployment flexibility. The use of industry leading technologies, careful procedural/object oriented software design and advanced software programming features ensure the software is robust, maintainable and scalable. Further testing of the prototype would have been greatly desirable, and will need to be completed if the project is taken further.

Overall, the TCAPS project has met it design goals, has a functioning prototype capable of demonstrating a majority of the features important to the TCAPS architecture and can therefore be considered a success.

It is the recommendation of this report that the TCAPS architecture is a feasible framework to commercially pursue for the provisioning of automated QoS over consumer broadband links. The current TCAPS prototype implementation met its design requirements, but falls short of commercial viability, and requires additional development work and testing to bring it up to a commercial level. The points discussed in section 9 of this report cover the additional work required before TCAPS could be publicly released. The estimated time frame for the completion of the critical additional work required for TCAPS commercial viability is 6 to 12 months.

Acknowledgments

This project would not have been possible without the assistance provided by Swinburne University's Centre for Advanced Internet Architectures and its director, Associate Professor Grenville Armitage.

Sincere thanks go to Grenville for his guidance, donated time throughout the year, and belief in my ability to complete this project.

I would like to thank CAIA's Dr. Philip Branch for his help in defining the Quake 3 traffic classification rule that was implemented as part of the TCAPS flow classifier module.

Thanks also go to CAIA's Mr. Sebastian Zander and Dr. Jason But for informal discussions on project sticking points throughout the year.

References

1. “Broadband Choice – Plan Search”, http://bc.whirlpool.net.au/bc-plan.cfm?state=vic&class=0&type=res&cost=60&pre=3000&conntype=1&conntype=4&conntype=6&conntype=5&speed=512&upspeed=0 (as at 25/11/2005)
2. T.Lang, G.Armitage, P.Branch, H-Y.Choo, "A Synthetic Traffic Model for Half Life", Australian Telecommunications Networks & Applications Conference 2003, (ATNAC 2003), Melbourne, Australia, December 2003, http://caia.swin.edu.au/pubs/ATNAC03/lang-armitage-branch-choo-ATNAC2003.pdf (as at 25/11/2005)
3. “Integrated Services in the Internet Architecture: an Overview”, http://www.ietf.org/rfc/rfc1633.txt (as at 25/11/2005)
4. “Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification”, http://www.ietf.org/rfc/rfc2205.txt (as at 25/11/2005)
5. “An Architecture for Differentiated Service”, http://www.ietf.org/rfc/rfc2475.txt (as at 29/04/2005)
6. “Multiprotocol Label Switching Architecture”, http://www.ietf.org/rfc/rfc3031.txt (as at 29/04/2005)
7. “Ubicom: Enabling ubiquitous communications”, http://ubicom.com/ (as at 25/11/2005)
8. Ubicom Inc., “Solving Performance Problems with Interactive Applications in a Broadband Environment using StreamEngine™ Technology”, October 2004, http://ubicom.com/pdfs/whitepapers/StreamEngine-WP-20041031.pdf (as at 25/11/2005)
9. “D-Link 108G Gaming Router”, http://games.dlink.com/products/?pid=370 (as at 25/11/2005)
10. “Bandwidth control, service level management”, http://www.allot.com/pages/solutions_content.asp?intGlobalId=26 (as at 25/11/2005)
11. “The FreeBSD Project”, http://www.freebsd.org/ (as at 25/11/2005)
12. “PF: The OpenBSD Packet Filter”, http://www.openbsd.org/faq/pf/ (as at 25/11/2005)
13. “ALTQ: Alternate Queueing for BSD UNIX”, http://www.csl.sony.co.jp/person/kjc/software.html#ALTQ (as at 25/11/2005)
14. “tcpdump/libpcap public repository”, http://www.tcpdump.org/ (as at 25/11/2005)
15. “The Adaptive Communication Environment”, http://www.cs.wustl.edu/~schmidt/ACE.html (as at 25/11/2005)
16. “Linux Advanced Routing & Traffic Control”, http://lartc.org/ (as at 25/11/2005)

Glossary

Term/Acronym	Definition/Description
API	**A**pplication **P**rogramming **I**nterface. It is the interface that enables one program to use facilities provided by another, whether by calling that program, or by being called by it
Bandwidth	Measurement of the speed of a link. Normally measured in bits per second
Bit	The most basic unit in digital electronics. Can have two states: on or off. In computing terms, is either a 1 for on, 0 for off
bps	**b**its **p**er **s**econd. Measures the speed of a digital communications link
Byte	A byte consists of 8 bits
BART	**B**roadband **A**ccess **R**esearch **T**estbed
Bridge	A piece of networking equipment that is used to physically segment networks and is responsible for passing inter-network traffic between the networks whilst leaving intra-network traffic on its originating network
BSD	**B**erkeley **S**oftware **D**istribution
CAIA	**C**entre for **A**dvanced **I**nternet **A**rchitectures
CI	**C**lient **I**nterface
CP	**C**ustomer **P**remises
CPE	**C**ustomer **P**remises **E**quipment
csh	The Berkeley UNIX **C** **sh**ell
DHCP	**D**ynamic **H**ost **C**onfiguration **P**rotocol
DiffServ	**Diff**erentiated **Serv**ices
FC	**F**low **C**lassifier
FIFO	**F**irst **I**n **F**irst **O**ut. Queue management technique where by elements that enter the queue first are the first to leave
FreeBSD	A free, UNIX based OS, developed by the University of California, Berkeley
FS	**F**low **S**ifter
GNU	**G**NU's **N**ot **U**nix
GPL	**G**eneral **P**ublic **L**icence. A free software licence from the GNU project.
HTTP	**H**yper**t**ext **T**ransfer **P**rotocol
IETF	**I**nternet **E**ngineering **T**ask **F**orce
IntServ	**Int**egrated **Serv**ices
IP	**I**nternet **P**rotocol
IPv4	See IP
IPv6	**I**nternet **P**rotocol **v**ersion 6. Next generation version of IPv4,
ISO	International Organisation for Standardisation
ISP	**I**nternet **S**ervice **P**rovider
Jitter	Variation in latency. Typically measured in milliseconds

Kbps	**K**ilo **bps**. 1000 bps i.e. 5kbps is 5000 bps
Latency	A measure of the amount of time it takes to send a single packet of data from one point in a network to another. Typically measured in milliseconds
LAN	**L**ocal **A**rea **N**etwork
Mbps	**M**ega **bps**. 1000000 bps i.e. 5mbps is 5000000 bps
ML	**M**achine **L**earning
MPLS	**M**ulti**p**rotocol **L**abel **S**witching
OEM	**O**riginal **E**quipment **M**anufacturer
OS	**O**perating **S**ystem
OSI	**O**pen **S**ystem **I**nterconnection. An ISO network communications standard
Packet	A unit of data sent across a network
Packet filtering	Selective passing or blocking of data packets as they pass through a network interface based on a defined set of rules
QoS	**Q**uality **o**f **S**ervice
Router	A device that determines the next network point to which a data packet should be forwarded enroute toward its destination
Shell	An operating system command interpreter
SS	**S**ub**s**ystem
TCP	**T**ransport **C**ontrol **P**rotocol. Connection oriented transport protocol
TX	Abbreviation for “transmit”
UDP	**U**niversal **D**atagram **P**rotocol. Connectionless transport protocol
UNIX	A cross platform operating system developed at AT&T labs
WWW	**W**orld **W**ide **W**eb

Appendices

TCAPS Project Final Report

Appendix A: Conference paper: “An Architecture for Automated Network Control of QoS over Consumer Broadband Links”

L.Stewart, G.Armitage, P.Branch, S.Zander, "**An Architecture for Automated Network Control of QoS over Consumer Broadband Links**", IEEE TENCON 05 Melbourne, Australia, 21 - 24 November, 2005