US 20070242675A1

(19) United States

(12) Patent Application Publication (10) Pub. No.: US 2007/0242675 A1

Romrell et al. (43) Pub. Date: Oct. 18, 2007

(54) DUAL SCHEDULING FOR EFFICIENT NETWORK TRAFFIC MANAGEMENT

(76) Inventors: **David Romrell**, Hillsboro, OR (US); **Christopher Charles Ptacek**, Beaverton, OR (US)

Correspondence Address:
OCCHIUTI ROHLICEK & TSAO, LLP
10 FAWCETT STREET
CAMBRIDGE, MA 02138 (US)

(21) Appl. No.: **11/404,049**

(22) Filed: **Apr. 13, 2006**

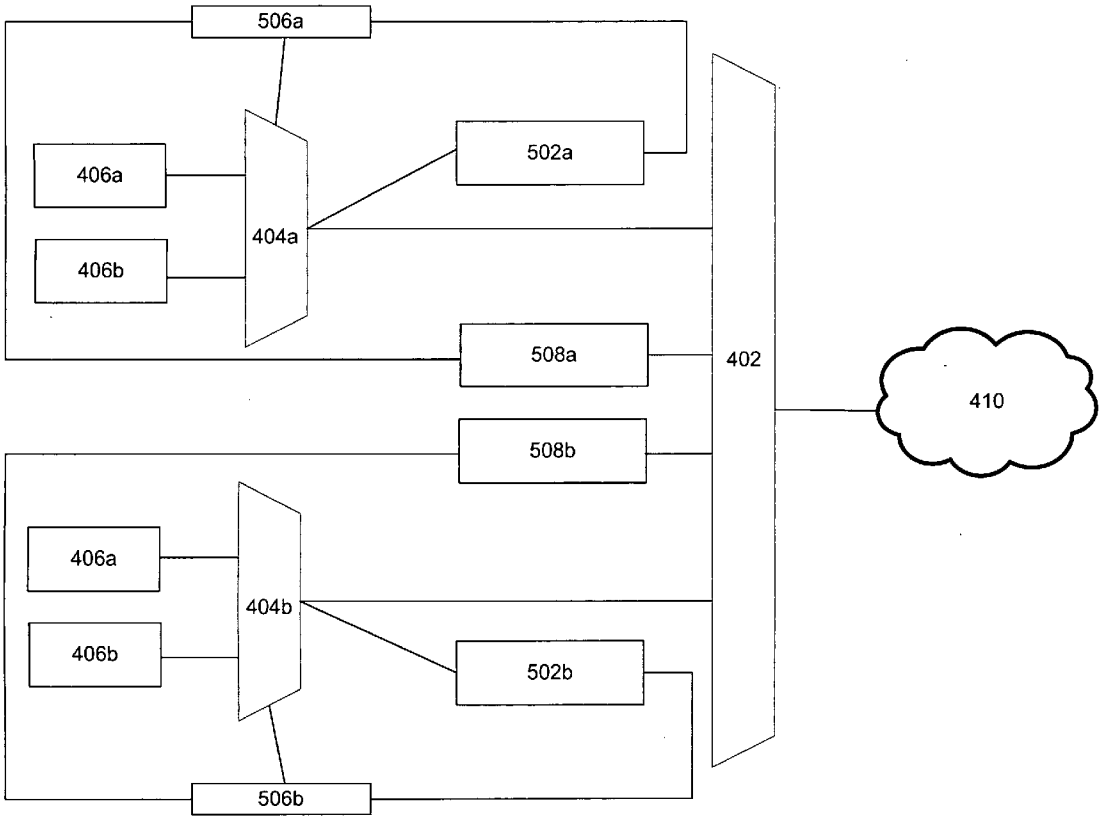
Publication Classification

(51) **Int. Cl.**
H04L 12/56 (2006.01)

(52) **U.S. Cl.** **370/395.4**

(57) **ABSTRACT**

Data traffic is scheduled by, in a first scheduler, selecting a source of traffic from a plurality of sources of traffic, each source being associated with a second scheduler, in a second scheduler associated with the selected source of traffic, selecting a type of traffic from a plurality of types of traffic within the source selected by the first scheduler, and transmitting data of the selected type and source.



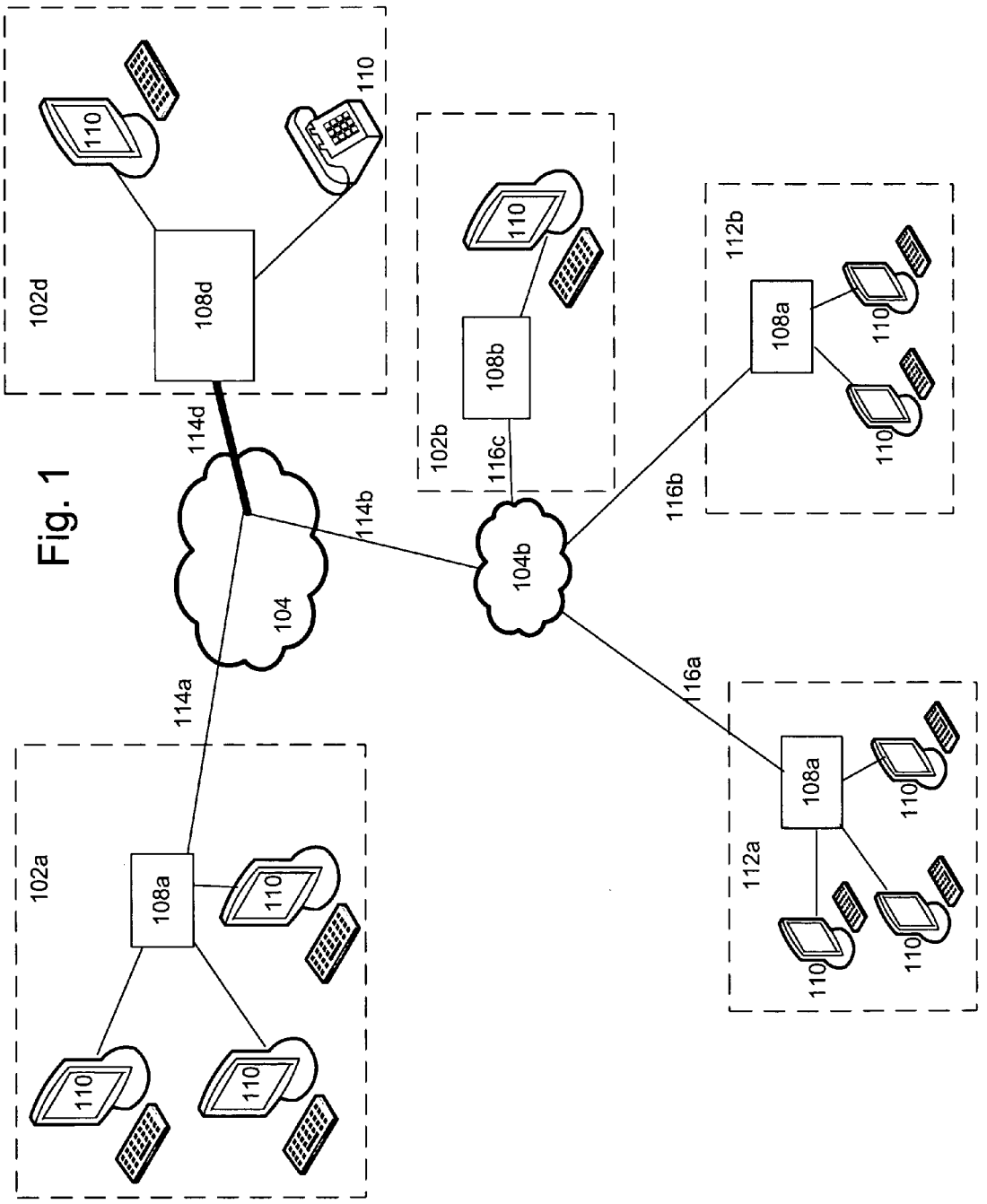

102d
108d
110
110
114d
104
114b
102b
108b
116c
110
104b
116b
112b
108a
110
110
114a
102a
108a
110
110
110
116a
112a
108a
110
110
110

Fig. 1

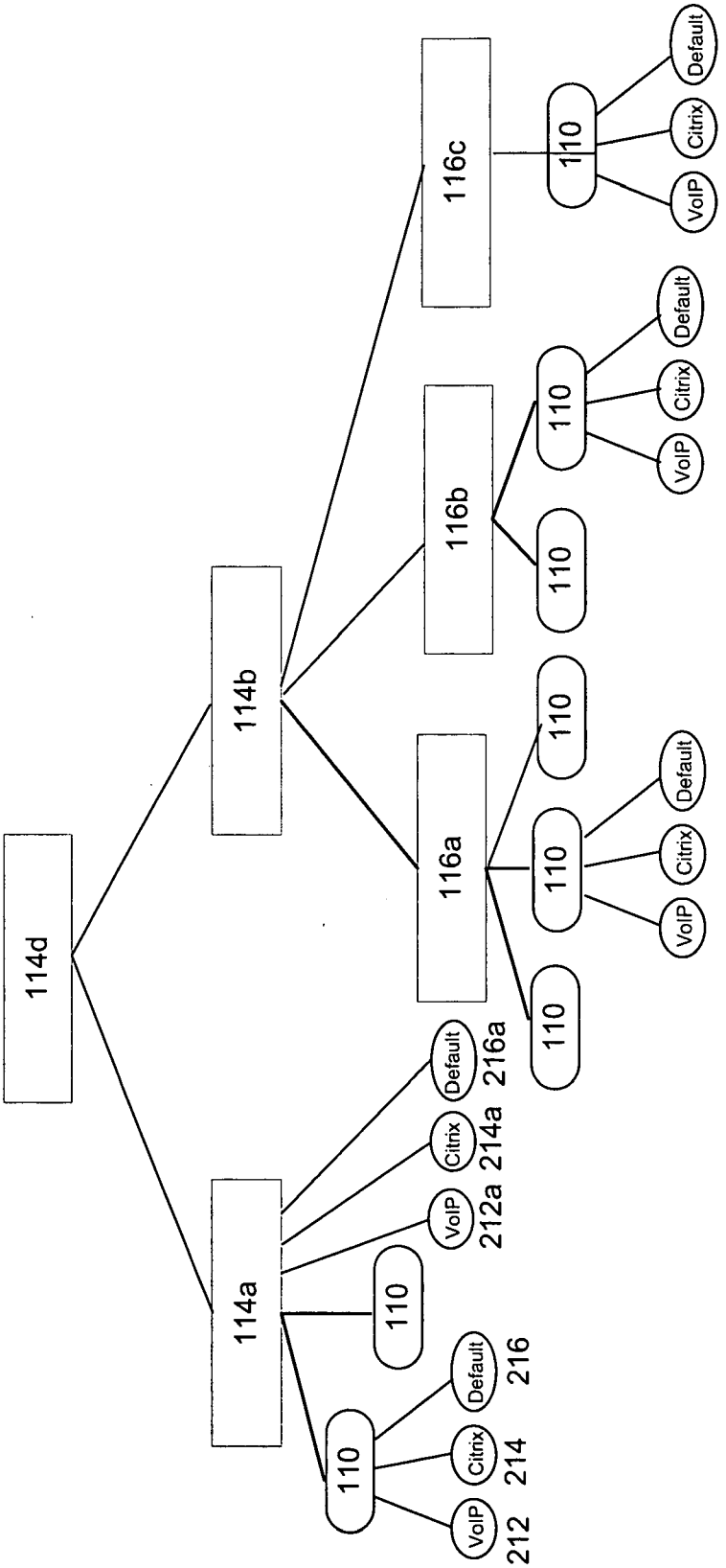

114d
114a
114b
116a
116b
116c
110
VoIP
Citrix
Default
212
214
216
212a
214a
216a

Fig. 2

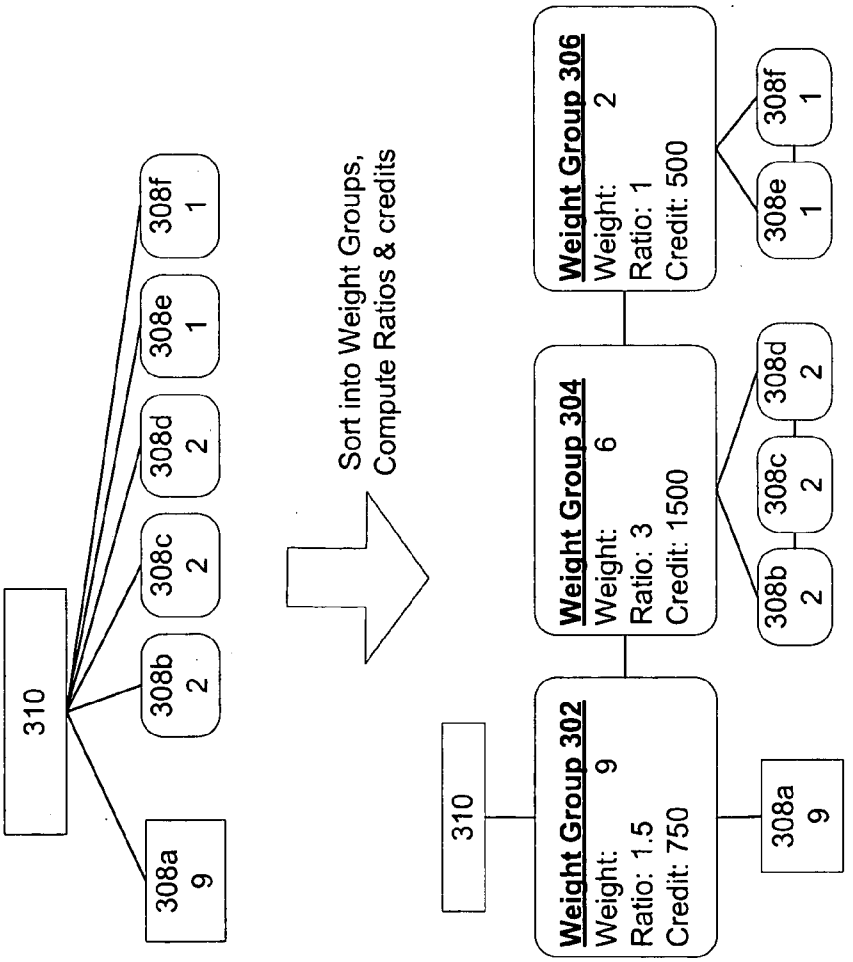

310
308a 9
308b 2
308c 2
308d 2
308e 1
308f 1
Sort into Weight Groups, Compute Ratios & credits
310
Weight Group 302
Weight: 9
Ratio: 1.5
Credit: 750
308a 9
Weight Group 304
Weight: 6
Ratio: 3
Credit: 1500
308b 2
308c 2
308d 2
Weight Group 306
Weight: 2
Ratio: 1
Credit: 500
308e 1
308f 1

Fig. 3

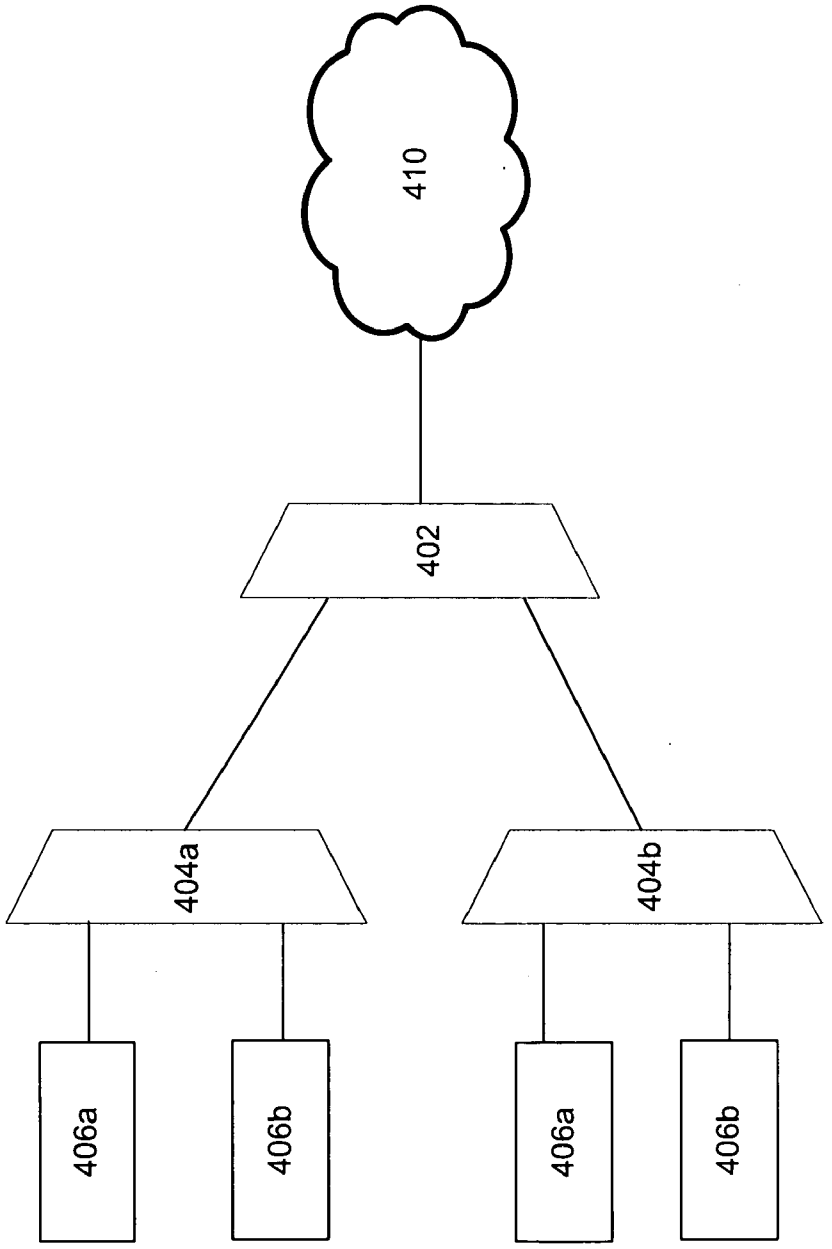


Fig. 4

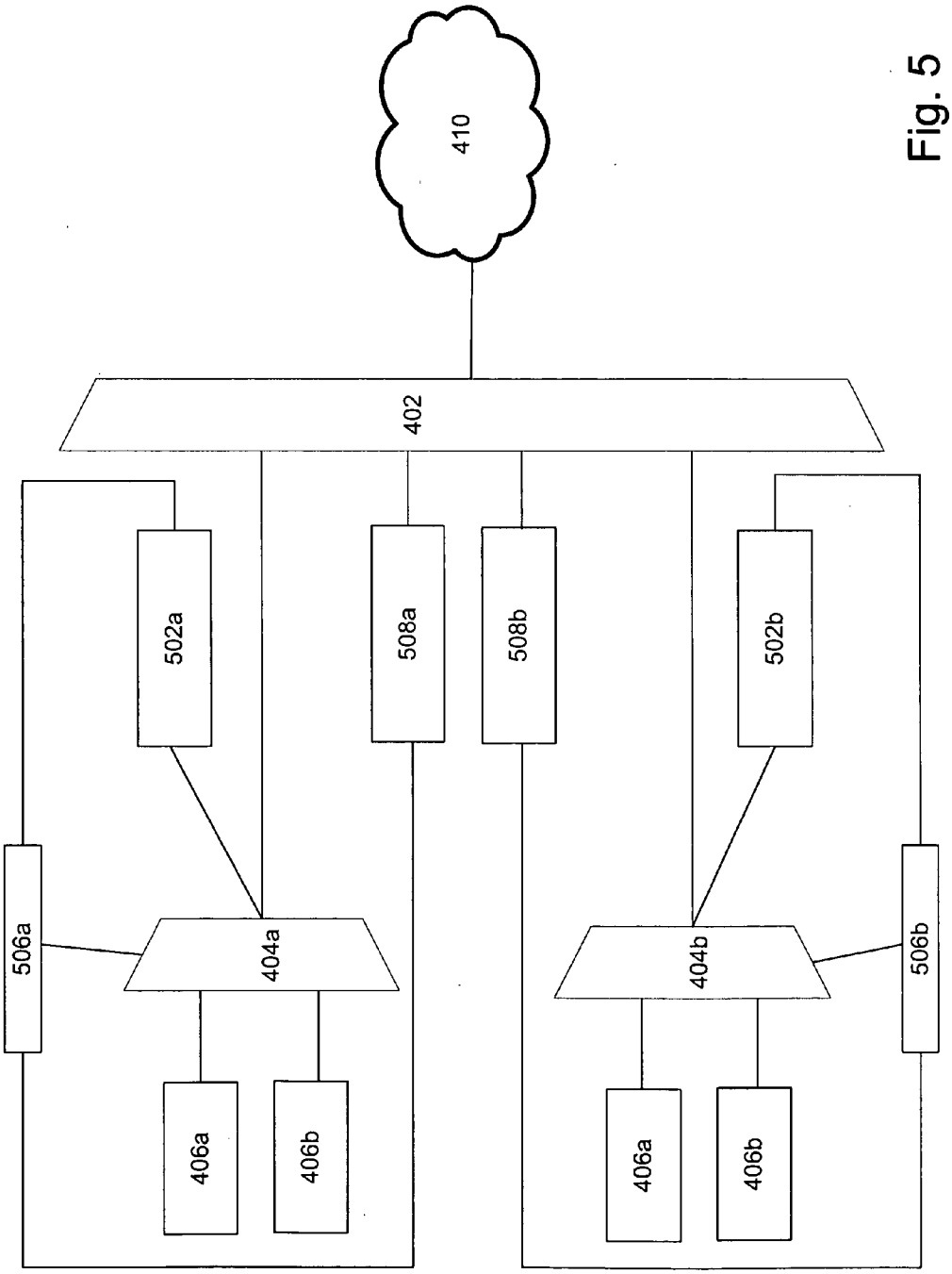

410
402
502a
508a
508b
502b
506a
404a
404b
506b
406a
406b
406a
406b

Fig. 5

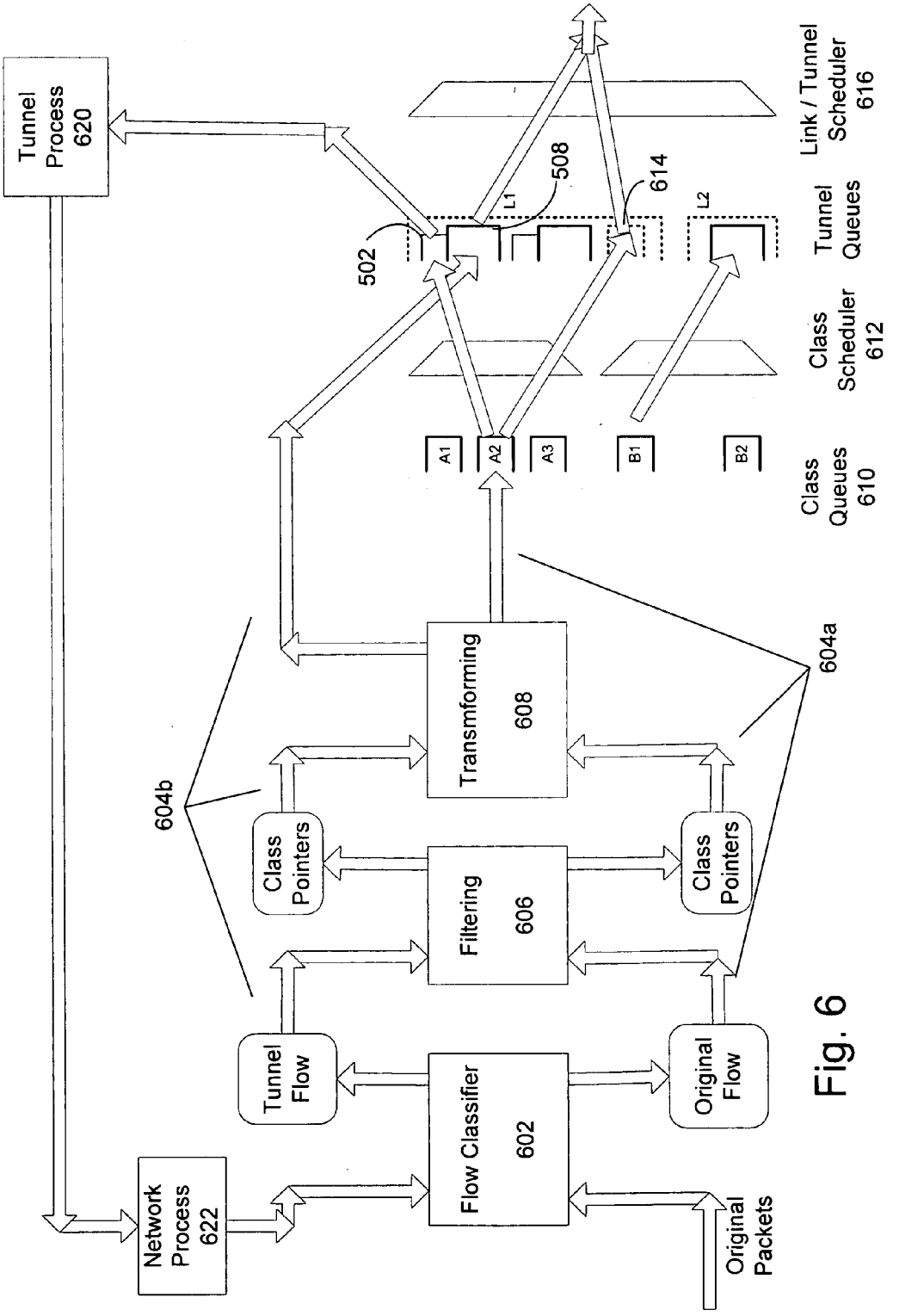

Tunnel Process 620
Link / Tunnel Scheduler 616
Tunnel Queues
Class Scheduler 612
Class Queues 610
502
508
614
L1
L2
A1
A2
A3
B1
B2
604a
604b
Transmforming 608
Class Pointers
Filtering 606
Class Pointers
Tunnel Flow
Original Flow
Flow Classifier 602
Network Process 622
Original Packets

Fig. 6

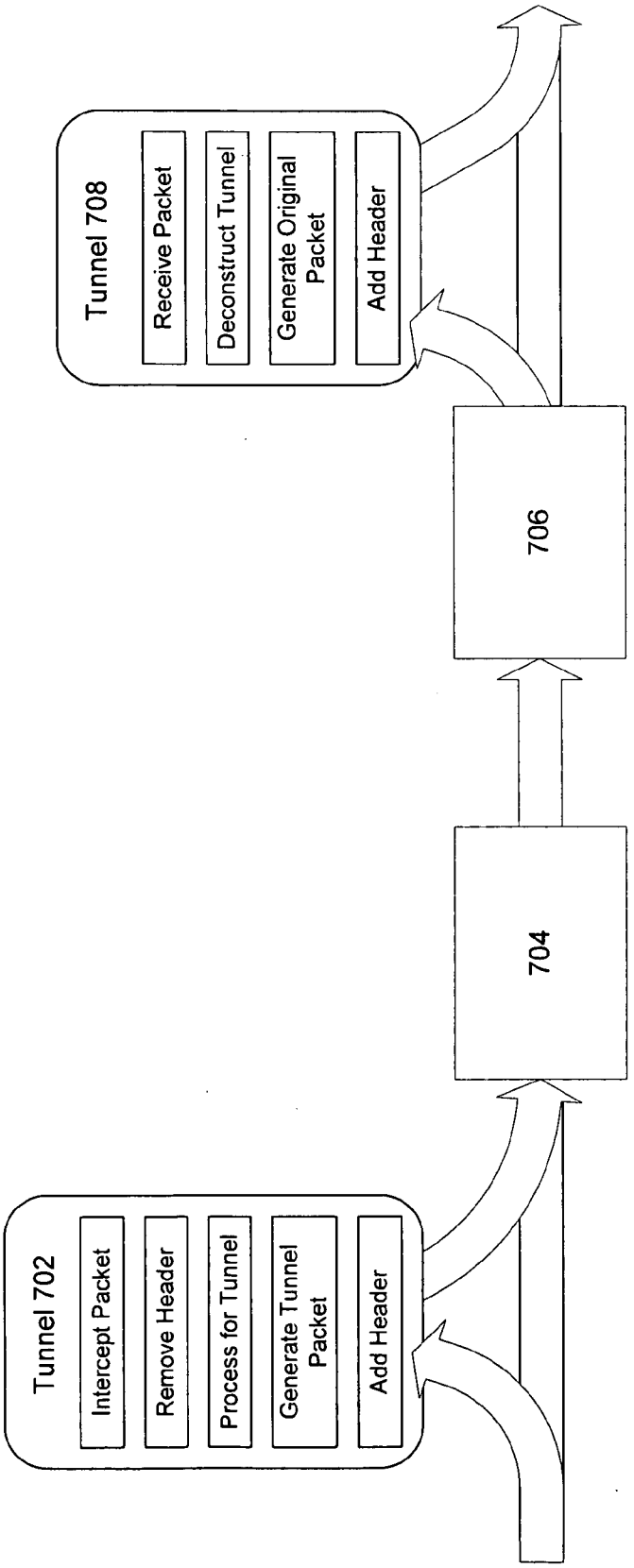

Tunnel 708
Receive Packet
Deconstruct Tunnel
Generate Original Packet
Add Header
706
704
Tunnel 702
Intercept Packet
Remove Header
Process for Tunnel
Generate Tunnel Packet
Add Header

Fig. 7

DUAL SCHEDULING FOR EFFICIENT NETWORK TRAFFIC MANAGEMENT

TECHNICAL FIELD

[0001] This description relates to dual scheduling for efficient network traffic management.

BACKGROUND

[0002] In operating a network, it is sometimes necessary to control the flow of data from one point to another. This is especially true in complex network topologies, such as a tiered structure as shown in FIG. **1**, with a central site **102***d* and several layers of sub-networks **102***a, b*, **112***a, b* each going through one or more links to reach the central site **102***d*. Previous systems for managing network traffic have relied on class based queuing (CBQ) or other scheduling systems to implement link level scheduling, that is, scheduling which of several links can send network traffic over an uplink to another tier of the network. Other systems have used data compression, requiring modifications to the systems at either end of a compressed link. Issues in scheduling network traffic include link oversubscription, where the various links into a node have a higher total traffic than the link out of the node to another part of the network, guaranteeing bandwidth amounts to various links and various classes of data traffic, and compensating for the effects of compression on allocation of bandwidth.

SUMMARY

[0003] In general, in one aspect, data traffic is scheduled by, in a first scheduler, selecting a source of traffic from a plurality of sources of traffic, each source being associated with a second scheduler, in a second scheduler associated with the selected source of traffic, selecting a type of traffic from a plurality of types of traffic within the source selected by the first scheduler, and transmitting data of the selected type and source.

[0004] Implementations include one or more of the following. Repeating the selecting and transmitting. The traffic is traffic for passing over a communications link. The selecting includes scheduling the selection of sources and types according to characteristics of the communications link. Selecting a source of traffic includes selecting a source from which packets should be delivered according to a rule. Delivering packets according to the rule includes one or more of guaranteeing a minimum bandwidth for a source of the plurality of sources, guaranteeing a maximum burst limit for a source of the plurality of sources, and guaranteeing a service interval to a source of the plurality of sources. Choosing a source of traffic includes allowing a user to configure a preemptive priority for a type of traffic. In the first scheduler, accounting for bandwidth used by each source of traffic. Selecting a type of traffic includes selecting a type from which packets should be delivered according to a rule. Delivering packets according to the rule includes one or more of guaranteeing a minimum bandwidth to a type, within an amount of bandwidth allocated by the first scheduler, guaranteeing a maximum burst limit to a type, within a burst limit allocated by the first scheduler, and guaranteeing a service interval to a type. The types of traffic include overlapping classifications of traffic. Before the selecting, filtering the traffic based on routes the traffic will use. The filtering includes applying a radix tree algorithm. Determining that a packet from the selected type is to be transmitted through a tunnel, and selecting a type includes charging the type for bandwidth usage based on an average efficiency of the tunnel.

[0005] In general, in one aspect, data traffic is scheduled by selecting a type of traffic, and determining that a packet from the selected type is to be transmitted through a tunnel, in which selecting the type includes charging the type for bandwidth usage based on an average efficiency of the tunnel.

[0006] Implementations may include one or more of the following features. Adding the selected packet to a queue for the tunnel. Extracting a packet from the queue for the tunnel based on one or more of efficiency of the tunnel, responsiveness of the tunnel, a maximum delay of the tunnel, and a minimum buffer of the tunnel. Compressing packets in the queue for the tunnel, updating an average compression ratio of the tunnel, and transmitting the compressed packets according to a scheduler that selects sources of traffic from a plurality of sources of traffic. Encrypting packets in the queue for the tunnel, updating an average expansion ratio of the encryption, and transmitting the encrypted packets according to a scheduler that selects sources of traffic from a plurality of sources of traffic. Selecting a type includes using a class-based queuing algorithm.

[0007] In general, in one aspect, data traffic is scheduled by selecting a source of traffic from a plurality of sources of traffic using a group ratio round robin scheduling algorithm.

[0008] Implementations may include one or more of the following features. Using a group ratio round robin scheduling algorithm includes defining an ordered set of groups of sources of traffic having similar weights, computing ratios between total weights of the groups, repeatedly, choosing one of the groups, within the chosen group, using a second algorithm to choose a source of traffic, transmitting an amount of traffic from the chosen source. The second algorithm is a deficit round robin scheduling algorithm. Computing a credit for each group based on the ratios, and after the transmitting, updating a deficit counter and a quantum counter for the chosen group based on the amount of traffic transmitted and the credit. Choosing one of the groups by, if the deficit counter and the quantum counter of the last-chosen group are above zero, choosing the last-chosen group, if the deficit counter of the last-chosen group is at or below zero, adding the credit to the deficit counter, adding a quantum to the quantum counter, and choosing the next group of the ordered set of groups, and if the deficit counter of the last-chosen group is above zero and the quantum counter is at or below zero, adding a quantum to the quantum counter for that group, and choosing the first group in the ordered set of groups.

[0009] In general, in one aspect, an amount of bandwidth to be used by a compression tunnel is determined by determining a pre-compression bandwidth limit for a type of traffic, determining a post-compression bandwidth limit for the tunnel, compressing data, including data from the type, determining a compression ratio, based on the compression ratio, determining an amount of pre-compression bandwidth used by the tunnel, and determining an amount of post-compression bandwidth used by the tunnel, and communicating the determined amounts to a scheduling process for

the type. For each of a plurality of types of traffic, guaranteeing an amount of bandwidth, determination of the guaranteed amount being based on the compression ratio. Scheduling types to use the tunnel based on the compression ratio and the determined amounts.

[0010] In general, in one aspect, for a plurality of endpoints of routes through a network, pairs of endpoints that can support tunnels to each other are identified, and identifications of the identified pairs of endpoints are recorded in a definition file. At each endpoint, the definition file is received, the identifications of other endpoints that the endpoint is paired with are read from the definition file, and a tunnel is created to each paired endpoint.

[0011] Implementations may include a centralized server performing the identifying and recording.

[0012] In general, in one aspect, packets to be transmitted are received. For each packet, a class and a link are identified, whether the packet should be transmitted using a tunnel is determined, the packet is added to a queue of packets having the same class as the packet, a class of packets is selected, packets from the selected class which are to be transmitted using the tunnel are added to a queue for the tunnel, the packets in the queue are adapted for the tunnel, producing adapted packets, adapted packets are added to a queue of packets to be transmitted on the link identified for the packets, a link is selected, and packets are transmitted from the queue for that link.

[0013] Implementations may include one or more of the following. Adapting the packets includes compressing the packets. Adapting the packets includes encrypting the packets. Adapting the packets includes encrypting and compressing the packets. Selecting a class of packets includes determining, for each class of packets, a number of bytes that have been compressed, a number of compressed bytes that have been transmitted, and a compression ratio, and selecting a class based on the compression ratio and the number of compressed bytes that have been transmitted for each class. Adapting the packets for the tunnel includes, for each packet, removing a network header from the packet, performing an operation on the packet to create an adapted packet, and adding a network header corresponding to a destination to the adapted packet. Receiving transmitted packets at the destination and for each packet that was transmitted using the tunnel, performing an inverse of the operation on the packet, adding a second network header to the packet, and transmitting the packet according to the second network header.

[0014] Advantages include the following. Bandwidth can be guaranteed to each branch in an oversubscribed network with thousands of links. Compression can be applied to a whole pipe or on a selective basis based on classes within a link. Certain types of traffic can be explicitly excluded from compression. The bandwidth used by a compression tunnel can be managed to a specified value. The bandwidth that a particular application or class of applications uses can be controlled to be within a specified range.

[0015] The details of one or more embodiments of the invention are set forth in the accompanying drawings and the description below. Other features, objects, and advantages of the invention will be apparent from the description and drawings, and from the claims.

DESCRIPTION OF DRAWINGS

[0016] FIGS. **1** and **2** are block diagrams of a network.

[0017] FIG. **3** is a block diagram of a scheduling algorithm.

[0018] FIGS. **4**, **5**, and **6** are block diagrams of schedulers.

[0019] FIG. **7** is a block diagram of a tunnel process.

DETAILED DESCRIPTION

[0020] In a central site network, such as that shown in FIG. **1**, multiple remote sites **102***a, b* and a central site **102***d* each have a single connection **114***a, b, d*, referred to as a link, through a network **104**, such as the Internet or a private IP network. Each site has network hardware **108***a, b, d*, which facilitates connections between devices **110** and the network links **114***a, b, d*, respectively. The remote sites **102***a,b* may also have links **116***a, b* to additional remote sites **112***a, b* connected through another network **104***b*. In such a case, the link to the local network hardware is shown as another link **116***c*, sharing the link **114***b* back to the central site **102***d* with the other remote links **116***a, b*. Connections between endpoints on the network are referred to as links, which may differ from actual network connections. Link **1114***d* connecting the central site to the network may be a larger capacity link than the remote site links **114***a, b* which feed in to it, or it may be the same or even smaller capacity. Similarly, link **114***b* could have a higher or lower capacity than the sum of remote links **116***a-c*.

[0021] Another depiction of a network is shown in FIG. **2**. Viewed this way, central site link **114***d* is at the top of the hierarchy. The two remote site links **114***a, b* are represented by the first level rectangular boxes while local systems **110** at each remote site are represented by rounded boxes. Second level links **116***a, b* to the more remote sites **112***a, b* are connected through remote site link **114***b*. Classes of data traffic originating from the various systems **110** are represented by ovals, e.g., classes **212** for VoIP traffic, **214** for Citrix traffic, and **216** for all other network traffic. Classes are sometimes shown directly feeding into a link, rather than coming through a system **110**, e.g. classes **212***a*, **214***a*, **216***a* connected to link **114***a*. At each level of the hierarchy, a link that represents several links at the next level down is referred to as a link group. For example, the link **114***b* is a link group that carries traffic from the links **116***a* and **116***b* from the remote sites **112***a* and **112***b* to the central site **102***d* via link **114***d*, as well as traffic on link **116***c* from the system **110** local to site **102***b*.

[0022] Each link may have a minimum guaranteed bandwidth, that is, the network is configured to assure that the capacity on link **114***d* associated with traffic for other links **114***a, b*, **116***a, b, c* is allocated at least to a minimum configured rate for that link. Links may also be configured with an allowable burst limit, that is, a maximum rate of traffic that the link can generate at any one time. Link oversubscription occurs when the total bandwidth available or used by a set of links into a system or site exceeds the bandwidth available on that site's link to the next level of the network hierarchy. For example, if each of links **116***a, b* could allow 1 Mb/s, but the outgoing link **114***b* could only provide 1.5 Mb/s, the link **114***b* would be oversubscribed. With inadequate scheduling, one link may use too great a portion of the available uplink bandwidth, preventing

another link from achieving its guaranteed minimum rate. Conversely, if the upstream link has a larger capacity than all the downstream links, e.g., if link **114***b* had a capacity of 10 Mb/s in the previous example, it could carry too much traffic and overwhelm the downstream links **114***a, b* to the remote sites **102***a, b*. The same problems are present in routing traffic on remote site link **114***b* to and from second-level links **116***a, b*. A link scheduler manages the traffic over each link to prevent oversubscription or overflowing of links. Such a scheduler determines which downstream link's traffic shall be carried by the upstream link (in either direction) at any particular time according to a link scheduling algorithm. A single central site scheduler, e.g., at device **108***d*, may operate at the top level of the network, modeling bottlenecks at all levels of the network to assure that link scheduling at each level is compatible with each other level. For example, a central site scheduler will not send more traffic over link **114***d* that is ultimately destined for links **116***a* and **116***b* than those links can handle, even if intermediate link **114***b* could handle that much incoming traffic.

[**0023**] In addition to actual connections between devices, different classes of network traffic may have different guaranteed minimum rates and burst limits. For example, VoIP traffic **212** may have a higher minimum and a lower burst rate than Citrix traffic **214** or regular network traffic **216**. For traffic within a link, a class scheduler determines which actual data packets to transmit, based on their class and a class scheduling algorithm. A single scheduler or set of schedulers could be implemented at a high level of the hierarchy, and their scheduling determinations cascaded down to the classes of traffic at each remote site. As with link scheduling, class schedulers operate on traffic flowing in either direction. In some examples, certain classes may have preemptive priority, in which case they not only take priority within their link, but the link itself is temporarily given priority over other links to assure packets for that class are quickly transmitted. As the preemptive class and link are satisfied the scheduler updates normal usage counters for the class and link scheduling algorithms.

[**0024**] In some examples, as shown in FIG. **3**, a link scheduler uses a group ratio round robin (GRRR) algorithm to determine what order to schedule the links in. Link group **310** represents lower-level links **308***a-f*, each with a different weight based on a guaranteed or actual rate. The GRRR algorithm uses "weight groups"**302**, **304**, **306** to group together links **308***a-f* that have similar weights. A list is maintained of a small number of groups **302**, **304**, **306** of links **308***a-f* having similar weights. For example, the link group **308***a* is in a group **302** of its own because it has a weight of nine. The links **308***b-c* are in a second group **306** because they have the same weight, two. The links **308***e-f* likewise form a group **306** of links with weight one. The groups **302**, **304**, **306** then have total weights of 9, 6, and 2 for relative ratios of 1.5 (9:6), 3 (6:2), and 1, respectively.

[**0025**] Groups are selected to transmit packets based on the ratio of bandwidth needed by one group to the bandwidth needed by each other group. Each group transmits an amount of data determined by the algorithm in turn. Within each group, individual links are selected to transmit packets based on the deficit round robin (DRR) algorithm, in which individual links are selected based on the ratio of their traffic volume to that of the other links in the group.

[**0026**] The GRRR algorithm, as adapted to link scheduling, proceeds as follows, within the example of FIG. **3**. A simple weighted round robin algorithm would schedule links **308***a-f* as AAAAAAAAABBCCDDEF (where letters A-F correspond to links **308***a-f*). While this provides overall fair bandwidth sharing, it does not provide optimal service latency. To be specific, the link group **308***a* will get bursts of congestion that cause queuing and possibly sustained packet loss. The other links get jitter because they wait for **308***a* to exhaust its weighted portion of link group **310**.

[**0027**] The GRRR approach resolves this by spreading weights over an entire service frame. In the example above it will schedule these as: AABAACAADEAABAACAADF. To achieve this, each link or link group is sorted into weight groups **302**, **304**, **306** with other peers that have weights within the same factor of 2 (e.g., rates between 2 k to 2 k+1−1). The weight groups are then sorted based on their total weight (i.e., the sum of the weights of the included links and link groups). Then the ratio of the weight from one group to the next is calculated, and a credit is assigned based on the ratios.

[**0028**] A pointer is kept on the current weight group being serviced. Credits and counters are used to maintain the ratios between the weight groups. A deficit counter is used to determine when to move to the next weight group. A quantum counter is used to determine when the current weight group ratio is satisfied and move the process back to the start. On the next invocation of the scheduler, that weight group is serviced and the counters are decreased by the amount of data sent. In some examples, the quantum is defined as a power of 2 (e.g., 1024 bytes) to simplify ratios by using a shift operation.

[**0029**] The scheduler moves the pointer between weight groups using the following ordered rules after servicing the current weight group:

[**0030**] 1. If the deficit and quantum counters are both above zero, then the pointer stays with current weight group.

[**0031**] 2. If the deficit is at or below zero, then the deficit credit is added to the deficit counter and the quantum credit is added to the quantum counter and then the pointer moves to next weight group.

[**0032**] 3. If only the quantum is at or below zero, then the quantum credit is added to the quantum counter and the pointer moves to (or remains at) the first weight group.

[**0033**] Because items within a group have weights within a power of 2 of each other, the scheduler can use simple deficit round robin within each weight group and still maintain good fair service latency. Table 1 demonstrates each step for the above process (for simplicity, a quantum size of 500 is used with simple packet sizes). In each step, the highlighted group transmits a variable number of bytes and the new deficit is shown in the Def+ column. The current quantum amount and quantum deficit are shown in the Qu and Qu+ columns, respectively. The two rules above are repeatedly followed as the process moves from one row to the next. This results in the schedule identified above.

TABLE 1

	Weight Group 302 Clients: A = 9 Weight: 9 Ratio: 1.5 Credit: 750			Weight Group 304 Clients: B = 2, C = 2, D = 2 Weight: 6 Ratio: 3 Credit: 1500					Weight Group 306 Clients: E = 1, F = 1 Weight: 2 Ratio: 1 Credit: 500				
slot	Def	Def+	Sent	Def	Def+	Qu	Qu+	Sent	Def	Def+	Qu	Qu+	Sent
	750		A	1500		500		B	500		500		E
1	750	−50	A 800	1500		500		B	500		500		E
2	700		A	1500	1100	500	100	B 400	500		500		E
3	700		A	1100	600	100	−400	B 500	500		500		E
4	700	300	A 400	600		100		C	500		500		E
5	300	−400	A 700	600		100		C	500		500		E
6	350		A	600	100	100	−400	C 500	500		500		E
7	350	−450	A 800	100		100		D	500		500		E
8	300		A	100	−400	100	−400	D 500	500		500		E
9	300		A	1100		100		B	500	0	500	0	E 500
10	300	−300	A 600	1100		100		B	500		500		F
11	450		A	1100	1000	100	0	B 100	500		500		F
12	450	−250	A 700	1000		500		C	500		500		F
13	500		A	1000	500	500	0	C 500	500		500		F
14	500	0	A 500	500		500		D	500		500		F
15	750		A	500	0	500	0	D 500	500		500		F
16	750		A	1500		500		B	500	300	500	300	F 200
17	750		A	1500		500		B	300	0	300	0	F 300

[0034] In some examples, a network includes thousands of links, but there will generally only be 3-8 weight groups. In most cases, most of the links will have similar rates (e.g., a typical network may have 400 links at 256 kb/s, **500** links at 512 kb/s, **8** links at 1.5 Mb/s, and 2 links at 3 Mb/s). Since weight groups are defined as weights within a power of 2, there are a maximum of 32 groups possible to cover all link types between 1 b/s (2^0) and 2 Gb/s (2^{31}). In other words, adding the GRRR mechanism to a link scheduler requires minimal memory overhead, as it requires minimal processing time while providing a very good ability to guarantee rates and fair service latency. Such a link scheduler is referred to as order O(1), meaning that the amount of computation necessary to operate it is substantially insensitive to the number of links being scheduled. The original GRRR algorithms were designed for process scheduling, and assume work units of fixed size, an assumption that is not necessarily true for packet schedulers.

[0035] By adding deficits as described above, each group has the to exceed its ratio during one transmission (e.g., the deficit of −50 in step 1) but this will decrease the volume of data that group can send by that amount the next time it comes to be serviced. This error is bounded by the maximum packet size or quantum size (whichever of the two is smaller) per service interval.

[0036] The variable size of packets is also the reason for the addition of the "quantum" measurements into the weighted group scheduler. This ensures that groups sending small packets will still get their fair ratio of bandwidth. As a weight group is serviced, the scheduler maintains the quantum deficit to assure the previous weight group ratio is satisfied. When the algorithm is moved to a new weight group, it is recharged with a new quantum (e.g., another 500 bytes is added). Any excess or surplus is taken into account during the next quantum (i.e., it is slightly less then it would normally be). The size 500 was used for simplicity of illustration. A size of 1024 is often used and is significant because it allows for efficient multiplication and division (by a simple bitwise shift left or right). This calculation of transition credits when weight groups are created or adjusted accounts for links becoming active/idle. In some examples, a quantum of 1024 bytes provides a good point in tuning the performance of the scheduler between efficiency and precision. A smaller quantum (e.g., 512 bytes) may have less error because it finds the best group to service next. This can help ensure the best service interval for links with small packet sizes (e.g., mostly voice or MTU limited traffic). However, this may come at the expense of efficiency in looping through groups until the quantum is large enough for one to send. Other O(1) packet schedulers use a quantum of the maximum packet size (e.g., 1600 bytes). The error introduced from the quantum is bounded to be less then a quantum difference per service interval.

[0037] In some examples, the GRRR algorithm assumes that all links within each weight group are active. If a group contains idle or over-limit links then the unused bandwidth from each such link would be given up by that link for the current service frame. It does this by assuming it did its DRR credit amount of work. The work is updated in both the DRR and GRRR deficits and the algorithm continues as if it was actual work done. This efficiently distributes the idle work done across all links and link groups. Each weight group maintains a list of active links and idle links. As a first packet is queued into an idle link, the link is moved to the tail of a DRR list and the weight is increased for the weight group. If not already set, then a pointer is set to the current location in the DRR list. Despite becoming active, the deficits are still tracked from before (i.e., they are not reset). This ensures that links that oscillate from active to inactive are not allowed to cheat at bandwidth. As the scheduler exhausts a link, (removes the last packet) it continues to leave it in the DRR until the next round. As the scheduler visits a link that is still exhausted on a second pass, it will then remove it from the DRR active list and put it into the idle list. It will also update the total weight of the weight group.

[0038] At the end of the service frame, the scheduler recalculates the ratios and credits for the effected groups. If the group does not shift in its sorted location then only the credit for the current group and the one that has larger weight needs to be updated. In some cases the order of the list may change because the adjusted weights for this group cause it to exceed the total weight of group in front of or behind it in the original list. In this case the credit needs to be updated on three groups (the two listed in the previous paragraph, plus the one above the previous location since its ratio is now to the group that had been behind us).

[0039] In some examples, this algorithm is performed even less frequently (i.e., every N times through the DRR cycle, or for multiple GRRR frames). On average the penalty for a delay before increasing the credit for new activation should be balanced by a similar delay before decreasing credit for idle link removal. The effect of a slightly low credit is a missed frame for the group but the per-item error is distributed between the items so they only loose a bit on the service latency. Conversely, when the credit is slightly high, the group may be given an extra slot in the frame, and this is also distributed as a slight boost to the service latency per link.

[0040] In some cases a weight group will only have one or two links within it. If those links go idle then the entire weight group should not take any slots from the frame. To do this the weight group is flagged as idle and will give up its slot by crediting either its quantum or next weight group credit (whichever is smaller). This maintains the ratio of work done with other weight groups. At the end of the frame this weight group is removed. The ratios and credits from the group in front of where it had resided are recalculated. When the link within the weight group becomes activated again, the weight group is moved back into the correct location and adjusts the ratios/credits for the group in front of it and for itself.

[0041] The above examples were of a single link group. This will be typical of most branch policies that will have root level link group with one or two links (e.g., a link to headquarters and a link to the Internet). In some examples, like that shown in FIG. **2**, there are policies that can have nested link groups in a hierarchy (below **114***b*). As shown in the above example, the link group **308***a* was given a fractional guarantee in the same way that links are given fraction guarantee. Within this link group, the central scheduler adds another GRRR scheduler to manage all of its children. The schedulers are run independently of each other but ensure precise service intervals for all items within the overall scheduler.

[0042] The weight groups and DRR algorithms provide fair bandwidth sharing and fair service latency based on guarantees. However, it has no concept of rate limiting which is a requirement, in some implementations, to represent the physical capacity of a link (so as not to exceed its rate, causing congestion). The rate limiting is done in a similar manner to class based queuing (CBQ). Each link and link group object is calculated to have an average time spent per byte for when it is running at its limit. Using this the scheduler tracks the next time to send, and an "average idle time" variable tracks the variance with the actual next time data is sent. If the next time is past (or not set) then the link or link group is not rate limited and can send. Otherwise, it has exceeded its rate and is skipped.

[0043] Within a selected link, a class scheduler is used to determine which data packets actually get transmitted over the link. Packets may be classified based on the application that generated them, priorities assigned by an application, or other factors. CBQ is one algorithm for scheduling traffic based on class. In CBQ, packets are scheduled according to relative priorities based on the type of data represented. For example, VoIP data needs low latency, while regular IP traffic can tolerate reduced latency but may require higher accuracy. In such an example, VoIP packets would be scheduled to be transmitted frequently and promptly, but not in large clusters. This sort of scheduling is greater than order O(1), meaning that the amount of computation necessary to operate a scheduler varies linearly with the number of classes, which may not be manageable for large networks.

[0044] Link-based scheduling and class-based scheduling can be combined as shown in FIG. **4** to achieve benefits of each without requiring burdensome amounts computation resources. A link scheduler **402** is used to select which link to allocate capacity to, but doesn't actually queue traffic to be transmitted. Rather, it simply selects which class scheduler **404***a, b* (there being one for each link) to take traffic from. The selected class scheduler **404***a* or **404***b* then selects packets from classes **406***a, b* and delivers them to the link scheduler to be transmitted. The link scheduler transmits packets provided by the class schedulers into the network **410**, for example, by sending them to a network interface of the machine on which the scheduler is operating. This process may be repeated at each stage of a hierarchical network like that shown in FIG. **2**, or may be done centrally and communicated to the responsible hardware at each site.

[0045] The typical packet filter to determine the class queue for a packet can be based on many packet attributes (address, port, type of service, packet flags, etc). However, mixing these filtering attributes allows filters to overlap so they are stored and searched in precedence order, which is O(N). On networks containing hundreds or thousands of links with many classes per link this is not generally scalable. The packet filtering shown in **606** of FIG. **6** uses a route based pre-filtering based on the link subnet definition to determine the link a packet will use. This pre-filtering can use routing algorithms like radix tree to allow an O(log(N)) search. A link only has a few classes, so within the link a normal O(N) precedence search can be done on rules to select the class a packet within the link should use. The class and link determination is then cached as part of a flow table, as disclosed in U.S. Pat. No. 7,010,611, which is incorporated here by reference, so when scheduling future packets the scheduler can do a quick hash to find flow and the previous class and link determination. In some cases this allows O(1) class and link determination.

[0046] Another method of regulating network traffic is to route some traffic through "tunnels." Tunnels are virtual connections between endpoints that operate within the actual network topology and that may provide additional features not provided by the network topology itself. Two common types of additional features provided by tunnels are compression and encryption. In a compression tunnel, packets are compressed so that they take up less bandwidth over the actual link used to transmit them. In an encryption tunnel, packets are encrypted so that network elements and users not privy to the encryption tools cannot read or understand intercepted data. Encryption tends to increase the size of

packets, and consequently the amount of bandwidth needed to transmit them. In both cases, packets are typically reconfigured (compressed or encrypted) for the tunnel at a stage prior to transmission, producing new packets which are then transmitted, and the process is reversed at the receiving end. In some examples, compression and encryption can be combined in a single tunnel. Both compression and encryption tunnels can complicate network management, since schedulers like those described above may be unable to determine the actual volume of data being transmitted or what class packets belong to. This may lead to inequitable scheduling, for example treating a particular link as if it had exceeded its share of bandwidth, when in fact all its traffic had been compressed after it was scheduled. In the alternative, a scheduler located downstream from the tunnel can take into account actual bandwidth used, but packets are obscured by the tunnel and accounting can't be based on the original classes of the packets.

[0047] To address these limitations, the combined scheduler discussed above may be combined with a feedback loop, as shown in FIG. **5**. Class schedulers sort incoming packets according to both their classification and whether they should go through a tunnel. Packets going through the tunnel are marked as such but they are still scheduled according to their classification. When selected by the class scheduler, they are placed in a queue **502***a, b* of packets destined for the tunnel. The tunnel processor **506***a, b* extracts packets from the queues **502***a, b*. This may include a series of packets based on minimum response time, maximum buffer, or other responsiveness controls. The queued packets are modified by the tunnel processor **506***a, b*, as appropriate to the type of tunnel, for example, they are compressed or encrypted, generating tunnel packets ready to be transmitted. By queuing packets and processing them together rather than compressing or encrypting them one at a time as they come through, greater compression efficiencies can be realized. This also avoids wasting bandwidth or CPU time when tunnel demands exceed capacity, and the resulting congestion can be managed by the schedulers. In other cases the tunnel processing may be performed on per-packet invocations without queues **502***a, b* and **508***a, b* and are then immediately transmitted onto the network **410**.

[0048] The compression ratio (or equivalently, a data expansion ratio) or other measure of tunnel performance that was achieved by the tunnel processes **506***a, b* is fed back to the class schedulers **404***a, b*, so they can determine an estimate of the bandwidth that will actually be used by the packets of each class, despite the fact that the original class of the packets may not be recognized when they are ultimately transmitted. The tunnel packets carrying the originally sent data packets are then returned to the beginning of the process, where they are identified as having already been processed for the tunnel. These packets then skip the class scheduler, since the original packets which make up the tunnel packets were already handled by that scheduler, and go directly to a queue **508***a, b* for the outgoing link scheduler. When the link scheduler next selects that link for transmission, it transmits both those packets in the post-tunnel queue **508***a, b*, and those provided to it by the class scheduler **404***a, b*. Links are scheduled based on the actual bandwidth they use, after compression or encryption of their tunnel packets, rather than on the amount of bandwidth they appeared to need before the packets were processed for the tunnel. Classes, meanwhile, are scheduled based on both the actual amount of raw data belonging to the class and the actual amount of post-tunnel-processing data that was transmitted.

[0049] Since tunnel packets are processed a second time by the link scheduler, it can treat them as their own queue, and by allocating a particular share of the corresponding link's bandwidth, the maximum bandwidth used by the tunnel can be controlled. Both the amount of data going into the tunnel and the amount of data actually used by the tunnel can be monitored and regulated. This process also allows classes to be set up prior to the tunnel and yet for the compression to be taken into account when scheduling particular classes. This may allow a class to exceed its allocated link bandwidth. For example, if a class is allocated a bandwidth of 1 Mb/s, but it put through a tunnel with a compression ratio of 2×, the class can be allowed to transmit 2 Mb/s without exceeding its allocation.

[0050] When the transmitted packets reach their destination, tunnel packets are processed to extract the original packets from which they were composed. These packets are then sent along to their ultimate destination, which may include processing through another set of schedulers and transmission over yet another uplink to another tier of the network.

[0051] Components of the schedulers can be reused for both pre-tunnel and post-tunnel packets, as shown in FIG. **6**. As packets enter the scheduling process, a flow classifier **602** classifies the packets and attaches a description of the packet including a reference to which flow (link, class, etc.) it was a part of. Both original packets and packets that have been through the tunnel processing once already go through the same flow classifier. The flow classifier and filtering determines whether a particular incoming packet should be routed through the tunnel, and includes that in its description. The fact that a packet has already been processed for the tunnel is another of the factors that goes into the classification and description of the packet. As packets move on to the next stage, they retain a reference to the flow they are a part of. Original packets are shown going through one pathway **604***a* and tunnel packets through another pathway **604***b*, though this distinction is merely for purposes of illustration. In a filtering stage **606**, packets are filtered according to their routes, classes, and whether they are to be excluded from the tunnel. Packets may be excluded from the tunnel because of the nature of the data or the priority of the application generating them, or because they have already been processed for the tunnel. For example, low-latency packets for VoIP may not be routed through a compression tunnel, because the time taken to compress the packets and route the tunnel packets back through the schedulers would violate the latency requirements of that class of data. Filtered packets retain their class identification as they pass on to the next stage.

[0052] Filtered packets next pass through a transforming stage **608** where additional actions may be performed, such as redirecting/hijacking packets to other processes, caching, rate-shaping, TOS (type of service) setting, imposing session limits, etc. This stage intercepts packets that have already been processed for the tunnel and directs them to the link scheduler, bypassing the class scheduler. New packets are queued in class queues **610** to be processed by the class scheduler **612**. The class scheduler **612** then directs packets

destined for the tunnel to a tunnel queue **502**. The tunnel process **620** will later extract these packets. Directing packets to the tunnel queue does not use the outbound link, so while the tunnel process is taking packets from the tunnel queue **502**, the link scheduler takes packets from the post-tunnel queue **508** and requests packets from the class scheduler **612** for the selected link (path **614**). Once it has sent the appropriate number of packets on to the outbound link, the link scheduler **616** moves on to the next link according to the scheduling algorithm. The link scheduler **616** may take into account whether a packet is a tunnel packet or a regular packet in selecting a packet to transmit on the network. For example, the link scheduler may be configured to interleave tunnel and non-tunnel packets to limit congestion within the network for each link being traversed. The scheduler can prevent a run of packets on an individual link from overwhelming that link.

[**0053**] Packets for the tunnel are queued as discussed above, processed by tunnel processor **620**, and returned to a network process **622**, for example, a TCP/IP, UDP, IP-TUN, IP-SEC, or SNP process, as if they were new packets to be transmitted. The network process **622** sends the packets to the flow classifier **602**. Since a single set of components can be used for both new and tunnel-processed packets, the feedback loop may be maintained without adding additional components. The class scheduler **612** tracks the amount of traffic for each class that it has directed to the tunnel, and receives feedback from the link scheduler **616** on how much bandwidth the tunnel packets actually used. This information is used by the class scheduler **612** to accurately account for the amount of bandwidth used by each class. Traffic can be managed based on the real classification of all packets, but each class accounted for based on the actual amount of data transmitted, including the tunnel savings.

[**0054**] This system has the advantage of a packet scheduler running prior to the tunnel so that it is allowed to manage traffic in classes before they are obscured, i.e., encrypted, or aggregated, i.e., compressed. For example, all database traffic can be scheduled at 10 Mb/s, and because of the compression tunnel, the total throughput can be up to 15 Mb/s, if either all traffic is compressed at a ratio of 1.5× or half the traffic is compressed at 2×. By sharing traffic management mechanisms, monitoring can be performed to show the relationship of pre-compression traffic usage and post-compression usage.

[**0055**] This system also has the advantage that if a tunnel was not established when network connections were initiated, or fails once established, packets destined for the tunnel can simply be transmitted like any other packet, and the class scheduler can schedule packets from each class according to the full amount of bandwidth used by each class.

[**0056**] In some examples, processing packets for inclusion in the tunnel includes the process shown in FIG. **7**. Packets identified for the tunnel are essentially hijacked out of their intended data pathway and routed into a tunnel process **702**. When packets enter the tunnel process **702**, they have a header which includes network routing information, such as the MAC (media access control) address of the next device **704** in the network that was expected to process the packet. Since the packet will be combined with others into a tunnel packet, and unpacked farther along the network to continue to its final destination, some of that routing information, e.g., the MAC address, has become moot and is removed from the original packet before the packet is processed and incorporated into a tunnel packet. A header is added to the tunnel packet, which directs the tunnel packet along the network and may be the same as the header that was removed from each of the constituent packets. At the receiving end **706** of the outbound link, a complementary process **708** to the outgoing tunnel process **702** unpacks the tunnel packets (for example, decompressing or decrypting them) and restores the original packets. At that point, the process returns the packets to the network kernel, which looks up new routing information, including the MAC address of the next machine each packet should be sent to, and uses that to create a new header. The packet is then sent on its way.

[**0057**] In some examples, tunnels are defined when the network is first initialized, and information about them is used in the above processes. A definition file is created which identifies all routes through the network, that is, all the pairs of machines that can transfer data between each other. This includes an identification of which pairs of machines should serve as endpoints of tunnels, for example, by compressing and decompressing some or all of the packets that travel between them, and the attributes of those tunnels. This definition file is communicated to all of the machines participating in routing traffic over the network. Each machine takes from this file the information that is relevant to it, such as identification of other machines to which it can create tunnels. The machine then sets up the tunnels expected of it, for example, by configuring the components necessary to process packets sent to the tunnel by the link scheduler in the processes described above. A centralized provisioning server can be provided to define all of the tunnels for a network and generate the definition file.

[**0058**] The techniques described herein can be implemented in digital electronic circuitry, or in computer hardware, firmware, software, or in combinations of them. The techniques can be implemented as a computer program product, i.e., a computer program tangibly embodied in an information carrier, e.g., in a machine-readable storage device or in a propagated signal, for execution by, or to control the operation of, data processing apparatus, e.g., a programmable processor, a computer, or multiple computers. A computer program can be written in any form of programming language, including compiled or interpreted languages, and it can be deployed in any form, including as a stand-alone program or as a module, component, subroutine, or other unit suitable for use in a computing environment. A computer program can be deployed to be executed on one computer or on multiple computers at one site or distributed across multiple sites and interconnected by a communication network.

[**0059**] Method steps of the techniques described herein can be performed by one or more programmable processors executing a computer program to perform functions of the invention by operating on input data and generating output. Method steps can also be performed by, and apparatus of the invention can be implemented as, special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application-specific integrated circuit). Modules can refer to portions of the computer program and/or the processor/special circuitry that implements that functionality.

[0060] Processors suitable for the execution of a computer program include, by way of example, both general and special purpose microprocessors, and any one or more processors of any kind of digital computer. Generally, a processor will receive instructions and data from a read-only memory or a random access memory or both. The essential elements of a computer are a processor for executing instructions and one or more memory devices for storing instructions and data. Generally, a computer will also include, or be operatively coupled to receive data from or transfer data to, or both, one or more mass storage devices for storing data, e.g., magnetic, magneto-optical disks, or optical disks. Information carriers suitable for embodying computer program instructions and data include all forms of non-volatile memory, including by way of example semiconductor memory devices, e.g., EPROM, EEPROM, and flash memory devices; magnetic disks, e.g., internal hard disks or removable disks; magneto-optical disks; and CD-ROM and DVD-ROM disks. The processor and the memory can be supplemented by, or incorporated in special purpose logic circuitry.

[0061] A number of embodiments of the invention have been described. Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the invention. For example, other network topologies may be used. Accordingly, other embodiments are within the scope of the following claims.

What is claimed is:

1. A method of scheduling data traffic comprising

(a) in a first scheduler, selecting a source of traffic from a plurality of sources of traffic, each source being associated with a second scheduler,

(b) in a second scheduler associated with the selected source of traffic, selecting a type of traffic from a plurality of types of traffic within the source selected by the first scheduler, and

(c) transmitting data of the selected type and source.

2. The method of claim 1 also comprising repeating steps (a)-(c).

3. The method of claim 1 in which the traffic is traffic for passing over a communications link.

4. The method of claim 3 in which the selecting comprises scheduling the selection of sources and types according to characteristics of the communications link.

5. The method of claim 1 in which selecting a source of traffic comprises selecting a source from which packets should be delivered according to a rule.

6. The method of claim 5 in which delivering packets according to the rule comprises one or more of

guaranteeing a minimum bandwidth for a source of the plurality of sources,

guaranteeing a maximum burst limit for a source of the plurality of sources, and

guaranteeing a service interval to a source of the plurality of sources.

7. The method of claim 1 in which choosing a source of traffic comprises allowing a user to configure a preemptive priority for a type of traffic.

8. The method of claim 1 also comprising in the first scheduler, accounting for bandwidth used by each source of traffic.

9. The method of claim 1 in which selecting a type of traffic comprises

selecting a type from which packets should be delivered according to a rule.

10. The method of claim 9 in which delivering packets according to the rule comprises one or more of

guaranteeing a minimum bandwidth to a type, within an amount of bandwidth allocated by the first scheduler,

guaranteeing a maximum burst limit to a type, within a burst limit allocated by the first scheduler, and

guaranteeing a service interval to a type.

11. The method of claim 1 in which the types of traffic comprise overlapping classifications of traffic.

12. The method of claim 1 also comprising, before the selecting, filtering the traffic based on routes the traffic will use.

13. The method of claim 12 in which the filtering comprises applying a radix tree algorithm.

14. The method of claim 1 also comprising

determining that a packet from the selected type is to be transmitted through a tunnel, and

in which selecting a type includes charging the type for bandwidth usage based on an average efficiency of the tunnel.

15. A method of scheduling data traffic comprising

selecting a type of traffic, and

determining that a packet from the selected type is to be transmitted through a tunnel,

in which selecting the type includes charging the type for bandwidth usage based on an average efficiency of the tunnel.

16. The method of claim 15 also comprising adding the selected packet to a queue for the tunnel.

17. The method of claim 16 also comprising extracting a packet from the queue for the tunnel based on one or more of

efficiency of the tunnel,

responsiveness of the tunnel,

a maximum delay of the tunnel, and

a minimum buffer of the tunnel.

18. The method of claim 15 also comprising

compressing packets in the queue for the tunnel,

updating an average compression ratio of the tunnel, and

transmitting the compressed packets according to a scheduler that selects sources of traffic from a plurality of sources of traffic.

19. The method of claim 15 also comprising

encrypting packets in the queue for the tunnel,

updating an average expansion ratio of the encryption, and

transmitting the encrypted packets according to a scheduler that selects sources of traffic from a plurality of sources of traffic.

20. The method of claim 1 in which selecting a type comprises

using a class-based queuing algorithm.

21. A method of scheduling data traffic comprising

selecting a source of traffic from a plurality of sources of traffic using a group ratio round robin scheduling algorithm.

22. The method of claim 21 in which using a group ratio round robin scheduling algorithm comprises

defining an ordered set of groups of sources of traffic having similar weights,

computing ratios between total weights of the groups,

repeatedly,

choosing one of the groups,

within the chosen group, using a second algorithm to choose a source of traffic,

transmitting an amount of traffic from the chosen source.

23. The method of claim 22 in which the second algorithm is a deficit round robin scheduling algorithm.

24. The method of claim 22 also comprising

computing a deficit credit and quantum credit for each group based on the ratios, and

after the transmitting, updating a deficit counter and a quantum counter for the chosen group based on the amount of traffic transmitted and the credits.

25. The method of claim 22 in which choosing one of the groups comprises

if the deficit counter and the quantum counter of the last-chosen group are above zero, choosing the last-chosen group,

if the deficit counter of the last-chosen group is at or below zero, adding the deficit credit to the deficit counter, adding a quantum credit to the quantum counter, and choosing the next group of the ordered set of groups, and

if the deficit counter of the last-chosen group is above zero and the quantum counter is at or below zero, adding a quantum credit to the quantum counter for that group, and choosing the first group in the ordered set of groups.

26. A computer-readable medium comprising instructions to repeatedly cause

a first scheduler to select a source of traffic from a plurality of sources of traffic, each source being associated with a second scheduler,

a second scheduler associated with the selected source of traffic to select a type of traffic from a plurality of types of traffic within traffic from the source selected by the first scheduler, and

data of the selected type and source to be transmitted.

27. A device for scheduling data traffic comprising

a first scheduler configured to select a source of traffic from a plurality of sources of traffic, each source being associated with a second scheduler, and

a second scheduler, associated with the selected source of traffic, configured to select a type of traffic from a plurality of types of traffic within traffic from the source selected by the first scheduler.

28. A method comprising

determining an amount of bandwidth to be used by a compression tunnel,

determining a pre-compression bandwidth limit for a type of traffic,

determining a post-compression bandwidth limit for the tunnel,

compressing data, including data from the type,

determining a compression ratio,

based on the compression ratio,

determining an amount of pre-compression bandwidth used by the tunnel, and

determining an amount of post-compression bandwidth used by the tunnel, and

communicating the determined amounts to a scheduling process for the type.

29. The method of claim 28 also comprising

for each of a plurality of types of traffic, guaranteeing an amount of bandwidth, determination of the guaranteed amount being based on the compression ratio.

30. The method of claim 28 also comprising scheduling types to use the tunnel based on the compression ratio and the determined amounts.

31. A computer-readable medium comprising instructions to cause a device to

determine an amount of bandwidth to be used by a compression tunnel,

determine a pre-compression bandwidth limit for a type of traffic,

determine a post-compression bandwidth limit for the tunnel,

compress data, including data from the type,

determining a compression ratio,

based on the compression ratio,

determine an amount of pre-compression bandwidth used by the tunnel, and

determine an amount of post-compression bandwidth used by the tunnel, and communicate the determined amounts to a scheduling process for the type.

32. A device for scheduling data traffic configured to

determine an amount of bandwidth to be used by a compression tunnel,

determine a pre-compression bandwidth limit for a type of traffic,

determine a post-compression bandwidth limit for the tunnel,

compress data, including data from the type,

determining a compression ratio,

based on the compression ratio,

determine an amount of pre-compression bandwidth used by the tunnel, and

determine an amount of post-compression bandwidth used by the tunnel, and communicate the determined amounts to a scheduling process for the type.

33. A method comprising

for a plurality of endpoints of routes through a network, identifying pairs of endpoints that can support tunnels to each other,

recording in a definition file identifications of the identified pairs of endpoints, and

at each endpoint,

receiving the definition file,

reading from the definition file the identifications of other endpoints that the endpoint is paired with, and

creating a tunnel to each paired endpoint.

34. The method of claim 33 in which the identifying and recording is performed by a centralized server.

35. A computer-readable medium comprising instructions to cause a device to

for a plurality of endpoints of routes through a network, identify pairs of endpoints that can support tunnels to each other,

record in a definition file identifications of the identified pairs of endpoints, and

at each endpoint,

receive the definition file,

read from the definition file the identifications of other endpoints that the endpoint is paired with, and

create a tunnel to each paired endpoint.

36. A device configured to

for a plurality of endpoints of routes through a network, identify pairs of endpoints that can support tunnels to each other,

record in a definition file identifications of the identified pairs of endpoints, and

at each endpoint,

receive the definition file,

read from the definition file the identifications of other endpoints that the endpoint is paired with, and

create a tunnel to each paired endpoint.

37. A method comprising

receiving data packets to be transmitted,

for each packet,

identifying a class and a link,

determining whether the packet should be transmitted using a tunnel, and

adding the packet to a queue of packets having the same class as the packet, selecting a class of packets,

adding packets from the selected class which are to be transmitted using the tunnel to a queue for the tunnel,

adapting the packets in the queue for the tunnel, producing adapted packets

adding adapted packets to a queue of packets to be transmitted on the link identified for the packets,

selecting a link, and

transmitting packets from the queue for that link.

38. The method of claim 37 in which adapting the packets comprises compressing the packets.

39. The method of claim 37 in which adapting the packets comprises encrypting the packets.

40. The method of claim 37 in which adapting the packets comprises encrypting and compressing the packets.

41. The method of claim 37 in which selecting a class of packets comprises

determining, for each class of packets,

a number of bytes that have been compressed,

a number of compressed bytes that have been transmitted, and

a compression ratio, and

selecting a class based on the compression ratio and the number of compressed bytes that have been transmitted for each class.

42. The method of claim 37 in which adapting the packets for the tunnel comprises, for each packet,

removing a network header from the packet,

performing an operation on the packet to create an adapted packet, and

adding a network header corresponding to a destination to the adapted packet,

the method also comprising

receiving transmitted packets at the destination, and

for each packet that was transmitted using the tunnel,

performing an inverse of the operation on the packet,

adding a second network header to the packet, and

transmitting the packet according to the second network header.

43. A computer-readable medium comprising instructions to cause a device to

receive data packets to be transmitted,

for each packet,

identify a class and a link,

determine whether the packet should be transmitted using a tunnel, and

add the packet to a queue of packets having the same class as the packet, select a class of packets,

add packets from the selected class which are to be transmitted using the tunnel to a queue for the tunnel,

adapt the packets in the queue for the tunnel, producing adapted packets

add adapted packets to a queue of packets to be transmitted on the link identified for the packets,

select a link, and

transmit packets from the queue for that link.

44. A device configured to

receive data packets to be transmitted,

for each packet,

identify a class and a link,

determine whether the packet should be transmitted using a tunnel, and

add the packet to a queue of packets having the same class as the packet, select a class of packets,

add packets from the selected class which are to be transmitted using the tunnel to a queue for the tunnel,

adapt the packets in the queue for the tunnel, producing adapted packets

add adapted packets to a queue of packets to be transmitted on the link identified for the packets,

select a link, and

transmit packets from the queue for that link.

* * * * *