Active-Active Firewall Cluster Support in OpenBSD

David Gwynne

School of Information Technology and Electrical Engineering, University of Queensland

Submitted for the degree of Bachelor of Information Technology

COMP4000 Special Topics Industry Project

February 2009

to leese, who puts up with this stuff

Acknowledgements

I would like to thank Peter Sutton for allowing me the opportunity to do this work as part of my studies at the University of Queensland.

A huge thanks must go to Ryan McBride for answering all my questions about pf and pfsync in general, and for the many hours working with me on this problem and helping me test and debug the code.

Thanks also go to Theo de Raadt, Claudio Jeker, Henning Brauer, and everyone else at the OpenBSD network hackathons who helped me through this.

Abstract

The OpenBSD UNIX-like operating system has developed several technologies that make it useful in the role of an IP router and packet filtering firewall. These technologies include support for several standard routing protocols such as BGP and OSPF, a high performance stateful IP packet filter called pf, shared IP address and fail-over support with CARP (Common Address Redundancy Protocol), and a protocol called pfsync for synchronisation of the firewalls state with firewalls over a network link. These technologies together allow the deployment of two or more computers to provide redundant and highly available routers on a network.

However, when performing stateful filtering of the TCP protocol with pf, the routers must be configured in an active-passive configuration due to the current semantics of pfsync. ie, one host filters and routes all the traffic until it fails, at which point the backup system takes over the active role. It is possible to configure these computers in an active-active configuration, but if a TCP session sends traffic over one of the firewalls and receives the other half of the connection via the other firewall, the TCP session simply stalls due to a combination of pfs stateful filtering and pfsync being too slow to cope.

This report documents an attempt at solving this problem by modifying the pfsync implementation and protocol to improve support of filtering OpenBSD routers in active-active configurations.

Introduction

OpenBSD is becoming increasingly popular as a platform for network security services, in particular network routing and firewalling. Currently there is support for building redundant firewalls with OpenBSD using pf and pfsync in active-passive configurations, however, to scale throughput between networks protected by OpenBSD firewalls it should be possible to configure them in active-active configurations. This project is an attempt at supporting active-active OpenBSD firewalls using pf and pfsync.

This document tries to give some context to the problems with building active-active firewalls with OpenBSD currently, then goes on to explain the changes that were made to enable it.

It assumes some familiarity with the C programming language, concepts relating to the workings of operating system kernels, and the IP network protocols including TCP.

Background

OpenBSD

OpenBSD is a general purpose UNIX-like operating system. As the name implies, it is descended from the long heritage of the Berkeley Software Distribution (BSD for short), which began as a set of modifications to the original AT&T UNIX and eventually grew into a completely independent code base.

The OpenBSD Project was created in late 1995 by Theo de Raadt who wanted to create a free and open source distribution with a focus on "portability, standardisation, correctness, proactive security, and integrated cryptography". While it is usable for a variety of tasks ranging from a desktop system up to file and print services, it is this focus on correctness and security that has let OpenBSD evolve into a highly usable and secure operating system for network security services. OpenBSD ships with a variety of software and capabilities in the base system that allow administrators to build very capable routers and firewalls.

OpenBSD As A Router

OpenBSD can be installed on a wide variety of hardware and with some simple configuration changes can be used as a standard router on IPv4 and IPv6 networks. It also includes support for technologies such as IPsec, Ethernet VLAN (802.1Q) tagging, a wide variety of network tunnelling protocols, a high quality firewall, IP and Ethernet load balancing, and can interoperate with other routers using protocols such as OSPF and BGP.

pf - The OpenBSD Packet Filter

For the majority of it's existence, OpenBSD has shipped with some form of packet filtering software that can be used to enforce network security policy at the IP and TCP/UDP layers for traffic flowing through an OpenBSD box. Originally the codebase included in OpenBSD was Darren Reeds IPFilter. However, in 2001 an issue with the license on that code caused it to be removed from the OpenBSD source tree.

Fortunately, shortly after its removal a developer named Daniel Hartmeier began implementing a new packet filter simply called pf. The code was quickly imported into the OpenBSD system, and development continued rapidly. Since that starting point pf has developed into an extremely capable and fast firewall.

The network policy that pf is to enforce is described in a ruleset which is parsed and loaded into pf inside the OpenBSD. These rulesets are generally just a list of pass or block rules that let you match against packets based on their address family, protocol, and if available their port number. However, pf is also able to perform actions such as normalising fragmented IP packets, and network address translation (rewriting IP addresses or port numbers inside a packet). Once loaded into the kernel the ruleset is then evaluated for every packet that enters or leaves the system. Packets going through an OpenBSD based router will be evaluated by pf twice, once coming into the network stack and again when it is leaving the stack.

Iterating over the ruleset for each and every packet going into the system can be slow though, but fortunately pf is a stateful firewall. This means that pf will try to keep track of connections going through itself. States use characteristics of the packet such as the address family, IP addresses, port numbers, and in the case of TCP, the state and sequence numbers of the connection. These states are organised into a red-black tree inside the kernel, and packets entering pf are compared against existing states before the ruleset is evaluated. If a packet matches a state, pf doesn't evaluate the ruleset and simply allows the packet though.

This means there are three benefits to the pf state table.

Firstly, there is an O(log n) cost for state lookups for a packet (where n is the number of states) vs an O(n) cost for a ruleset evaluation (where n is the number of pf rules). Since most packets will belong to an existing state they will therefore tend to match existing states, providing a huge speedup in the per packet evaluation costs.

Secondly, because pf does have some state about a connection it is able to independently check that a packet does in fact belong to an existing connection, and is not some spoofed packet with similar characteristics (ie, IP or port numbers).

Thirdly, because the state table will allow packets through for existing connections, the ruleset only has to concern itself with matching packets that create connections. This is another security benefit since without the state rules would have to be written that allow packets in both directions through a firewall. To highlight the difference, here is an example ruleset for TCP going through pf that does not take advantage of the state table:

```
pass in inet proto tcp from any port >1024 to $webserver port 80 no state
pass out inet proto tcp from $webserver port 80 to any port >1024 no state
```

As you can see this ruleset has to allow TCP replies from the web server back through the firewall otherwise the connection will fail. A ruleset taking advantage of the state table would look like this:

```
pass in inet proto tcp from any to $webserver port 80 flags S/SA
```

This rule only allows TCP packets with the SYN flag set (out of the SYN/ACK pair) to the webserver through the firewall. pf by default will create a state for this connection and will automatically allow replies just for that connection back through.

pf is quite scalable, meaning that with a suitable machine you can easily run firewalls with many network interfaces and with thousands of firewall rules. The number of states the firewall will manage is dependant largely on the amount of memory in the system. It is common for pf to be managing tens of thousands of concurrent state entries.

pfsync

While it has been established that stateful firewalling is good and a necessary thing for pf to operate at speed, it is unfortunately not a good thing in the event of a firewall failing. For simply routing packets it is trivial to configure two routers with the same configuration and in the event of one failing you can simply replace the failed unit with the functional one.

There are several systems in protocols to automate that failover, indeed, OpenBSD features several such as CARP (Common Address Redundancy Protocol), ospfd (a routing daemon implementing the OSPF protocol), and bgpd (a BGP implementation). These systems allow a router to take over from a failed peer within a matter of seconds.

However, if these routers are running pf then existing network connections through the redundant peer will no longer work since it has no knowledge of the state table the failed firewall had built. To compensate for that the pfsync protocol was developed.

pfsync simply sends messages between pf firewalls that notify each other other state inserts and deletions, and most importantly, state updates. Because pf keeps track of the TCP sequence numbers it is important for it to notify its peers when those sequence numbers progress so in the event of a failure the peer knows where the TCP connection was up to.

pf with pfsync allows the deployment of fully redundant stateful firewalls. In the event of failure the redundant firewall will have a full knowledge of the current state table and will be able to continue forwarding packets for those existing connections. However, because of a semantic in the implementation of the pfsync protocol, it is currently not possible to build clusters where both firewalls are actively able to handle traffic.

Active-Active Routers and Firewalls

It is becoming more common to deploy multiple routers between networks to increase the forwarding performance between those networks. Two routers are obviously twice as capable as a single router at forwarding traffic.

Since OpenBSD can be deployed as a firewall and router, it would be nice to be able to increase performance of such a deployment by simply adding firewalls as necessary. However, as stated above, this is currently not possible. Why this is the case is described in detail below.

OpenBSD Kernel Semantics

pf and pfsync both operate entirely within the OpenBSD kernel, only taking configuration or monitoring requests from userland. It is therefore necessary to understand some of the semantics of the OpenBSD kernel to effectively develop software in that context.

Firstly, despite the recent development for support of multiple processors in the OpenBSD kernel, it still largely operates as if were running on a single processor system. On a SMP system only one processor is able to be running the kernel at any point in time, a semantic which is enforced by a Big Giant Lock. There are portions of the kernel which are not protected by the Big Giant Lock, but they are not relevant to this discussion. The network stack and therefore pfsync still run under the Big Giant Lock.

There are two contexts for code execution in the kernel: interrupt context and process context. The differences between these two contexts affect how the locking of data structures and code are performed, and also affects which internal kernel APIs are available or appropriate for use.

The kernel has a process context when it has been asked to perform an action by a userland process via a system call. The OpenBSD kernel is non-preemptible, meaning that any code running in the kernel has to explicitly give up control before anything else in the kernel will run. If a critical path is only modified from a process context then simply having process context is sufficient to guarantee exclusive access to that section. However, if a critical path may sleep or yield control of the CPU, then additional locking is required. The OpenBSD kernel provides reader/writer locks for processes in the kernel to lock against each other.

The only exception to the kernel's non-preemptibility is for interrupts. Interrupt handlers may run at any time unless their interrupt source is masked and therefore prevented from running. Interrupt handlers are established at an interrupt priority level (IPL). To prevent interrupt handlers you simply mask interrupts at that level before executing your critical path. IPLs, as their name implies, are levelled. A high IPL prevents interrupt handlers established at lower priorities running at the same time. These IPLs range from IPL_NONE (no interrupts are masked) up to IPL_HIGH (all interrupts are masked). Interrupt handlers are run at their IPL, meaning that if you're currently executing code in a hardware network interrupt handler established at IPL_NET, you are guaranteed that no other interrupt handler at IPL_NET will run at the same time.

This ability to mask interrupts is the fundamental locking primitive in the OpenBSD kernel for data structures used in interrupt handlers. When a userland process enters the kernel, the effective interrupt mask level is at IPL_NONE, meaning no interrupts are masked at all. If this code wishes to modify structures that are also handled during interrupts, it must raise the IPL to prevent those interrupt handlers from running.

The only other large difference between interrupt and process contexts is that you cannot yield in an interrupt handler. The code must return, it is not able to sleep since sleeping relies on being able to switch from the current process to a new one.

The network stack in OpenBSD runs at two IPL levels: IPL_NET and IPL_SOFTNET. IPL_NET is used by drivers for network hardware and protects their data structures. The majority of the network stack runs at IPL_SOFTNET which is a lower priority than IPL_NET. Packets move between the network stack and the hardware via queues protected with IPL_NET. This allows packet processing to occur at the same time as packets are moved on and off the hardware. Packets may be received and queued for processing while processing of other packets is still in progress. It is therefore possible for many packets to be processed in a single call to the softnet handlers.

There are several softnet interrupt handlers implemented in the OpenBSD kernel, generally one per address family. Each of these handlers can be run from software by generating a software interrupt anywhere in the kernel. pf is run when the softnet interrupt handlers for the IPv4 and IPv6 protocols are run. Each handler de-queues a packet from the hardware passes it to pf for testing. If pf passes the packet it is then allowed to go down the rest of that particular address families stack, otherwise the packet is dropped. For a packet coming out of the stack, it is tested by pf before being put on an interfaces send queue. If pf makes a change to its state table because of that packet, it in turn notifies pfsync with that state. All this occurs at IPL_SOFTNET.

Another relevant detail about the OpenBSD kernel is that there are no readily usable high resolution timers that can be used to schedule events in the future with. The most usable timer has a resolution of 100 ticks a second. This is important to know if you're trying to mitigate some activity within the kernel.

pfsync In Depth

pfsync can be thought of as being made up of two parts: the protocol and the implementation.

The current version of the pfsync protocol is version 4. The protocol is surprisingly simple. Each message is an IP (the protocol allows either IPv4 or IPv6) encapsulated datagram that contains a small header prefixing a series of messages. The IP protocol ID for pfsync packets is 240, it does not get encapsulated inside a UDP or TCP. The header describes the type of these messages, and the number of them to expect in the packet. Each packet may only contain messages of the one type specified in the header, and it may only include as many messages as may be contained in a single IP datagram without needing to perform fragmentation. The size of the IP datagram is determined by the medium over which it is transmitted.

An exact knowledge of the layout of the header and the messages is generally unnecessary, however, some details are useful to know.

While a pf state may be uniquely identifiable by the characteristics of the packets it represents (ie, the address family, protocol, port numbers, etc), it can be inefficient to exchange these details between pfsync peers.

To address this inefficiency, an arbitrary state must be uniquely identifiable by all peers exchanging pfsync messages by a 96 bit key made up of a 32 bit "creator id" and a 64 bit "state id". Each peer in a pfsync cluster randomly generates a 32 bit creator id on boot which uniquely identifies that particular peer within the cluster while it is alive. Each state that the peer creates must be uniquely identifiable within the system by a state id. The combination of those two values allows pfsync to uniquely refer to any state amongst all its peers.

The message types within the pfsync protocol are:

PFSYNC_ACT_CLEAR	Clear pf state table A host will send this message when an administrator on the system has requested that the state table be flushed. On receiving this message the host must flush its state table.
PFSYNC_ACT_INS	Insert a new state into the state table When pf has processed a packet and decided to create state for it, it will request pfsync send a message of this type to describe that state to its peers A peer receiving this message creates a new state for the traffic described in the message and inserts into its local state table.
PFSYNC_ACT_UPD	Update a state When a packet causes a state to be modified, eg, a TCP packet will move the sequence numbers within the state along, it will generate a message of this type describing the state. A peer receiving this message attempts to locate an existing state and updates it with the new details. If no state has been found it may create and insert a new state with the specified details.
PFSYNC_ACT_DEL	Delete a state When a peer has received a packet that terminates a state, or a state times out, it will send a message of this type. A peer receiving this message will attempt to locate this state and remove it from its local state table.

PFSYNC_ACT_UPD_C	"compressed" state update This is semantically the same as the PFSYNC_ACT_UPD message, except instead of exchanging all the state details it only contains the creator and state ids to identify which state the update is for.
PFSYNC_ACT_DEL_C	"compressed" state deletion This is semantically the same as the PFSYNC_ACT_UPD message, except instead of exchanging all the state details it only contains the creator and state ids to identify which state to delete.
PFSYNC_ACT_INS_F	insert fragment pf may maintain a cache of fragmented packets and reassemble them to allow proper stateful filtering on those fragments. This message is intended to share the contents of the fragment cache between peers.
PFSYNC_ACT_DEL_F	delete fragment this request is intended to remove a fragment from the shared fragment cache.
PFSYNC_ACT_UREQ	request "uncompressed" state update If a pfsync peer receives a compressed update for a state it cannot locate by the creator and state ids, it can request an uncompressed state update so it can enter the current state details into its local table. A peer may request a a bulk update by sending a message with the creator and state ids set to 0. A peer receiving this message will search its local state table for an entry identified by the creator and state ids and will send a message of type PFSYNC_ACT_UPD in response. If the peer receives a request with the creator and state ids set to zero it will initiate a send of update messages for all of its states.
PFSYNC_ACT_BUS	bulk update status A peer will send this message after completing a bulk send of update messages. A peer receiving this message may be confident that it has a usable copy of the pf state table and may actively participate in the cluster.

PFSYNC_ACT_TDB_UPD	TDB replay counter update If the peer is acting as an IPsec gateway, it will send updates to the replay counters on the IPsec security associations to its peers. A peer receiving these messages will search for an IPsec security association with the same parameters and update the replay counters with the values provided by the pfsync message. This can be used to provide failover for IPsec gateways

pfsync doesn't recognise individual peers in the cluster, it only deals with communicating with the cluster as a whole. pfsync by default sends packets to a multicast group and receives packets destined for that multicast group. If a peer receives a compressed update it knows nothing about, it will send an update request to the multicast group, not to the specific peer the compressed update came from.

The current implementation of pfsync is implemented as a pseudo network interface driver. This is largely to allow for the creation and configuration of pfsync via ifconfig, which is a familiar administration tool for network services in UNIX-like systems. Despite being a network interface it does not provide and endpoint for traffic to be routed in or out of in the kernel.

A pfsync interface requires very minimal configuration to become usable. It is enough to define which interface the pfsync packets are to be exchanged on, and then bring the pfsync interface up. If pf is active on the system then it will begin to notify pfsync of inserts, updates, deletions, and clears of states in the pf state table.

Internally pfsync builds pfsync packets on the fly when pf issues notifications to it. The first notification from pf will cause pfsync to allocate memory for a packet to be built in, and will write the message for that notification into the packet. A second notification from pf will cause pfsync to search that packet it is building for a message about that same state. If the state was found in the packet already, the message is updated with the new parameters, otherwise a new message is added to the packet about the state.

pfsync will eventually transmit the packet it is building on several events.

The first is from a timeout that is scheduled one second from when the packet was first allocated. When that timeout is hit the packet is sent out with whatever updates were made during that 1 second.

The second condition is if any particular state described in a pfsync packet is updated more than a certain number of times. The maximum number of updates a state in a packet may receive is 128 by default, but that value is configurable by the user.

Both of these conditions are there to try and mitigate against pfsync generating a single packet for each and every state update that pf notifies it of. Without this mitigation the pfsync traffic between peers in a cluster could conceivably exceed the traffic pf is firewalling.

The third condition a packet is transmitted on is caused when pfsync has to switch message types for the packet is building. Because pfsync packets may only contain messages of one type, if pfsync is notified of a different type of action to the one it was building a packet for it will simply transmit the current packet immediately. It will then allocate a new packet for messages of the new type and will start filling it in again.

The fourth condition a packet will be sent out on is if it is in response to an action that requires peers know about the change immediately. For example, if pf notifies pfsync of a state table clear, pfsync will build that packet and send it immediately. Also, if a peer requests an uncompressed update about a state, that message is also built and sent out immediately.

Lastly, if an additional message will cause the packet to grow larger than the MTU of the physical interface specified for the pfsync traffic, the current packet will be transmitted immediately.

When a pfsync packet is received by the network stack, it is sent immediately to the pfsync input routine. The input routine simply iterates over the messages that were specified by the header and takes action accordingly. Generally the implementation follows the actions that the protocol specifies that were described above. However, some aspects of the implementation of the pfsync receive code are smarter about their actions than what the protocol would imply.

For example, iot is possible that a host will receive an update for a state that has been modified locally as well as by another peer, which is quite likely when you are moving from a passive to active role in a cluster. In that situation it will try to take the most current details from the state and the update it just received and merge them together. If it has determined that the peers version is stale, it will immediately send an update of the state based on its merged information.

Also, pfsync attempts to keep track of how current it thinks it is with respect to its peers, and it feeds that state back into the system. When the pfsync interface is first brought up it marks itself as "not ok" and sends out a request for a bulk update by transmitting a PFSYNC_ACT_UREQ with 0s set for the creator and state ids. It is only after it has successfully received a PFSYNC_ACT_BUS message that it will move itself to "ok". If the firewall is using carp as part of an active-passive cluster, the pfsync ok state is used to demote the carp interfaces so it will not receive traffic until it is ready to continue forwarding traffic with the existing states. If pfsync does not receive such a message, it will request a bulk update another 12 until it times out and moves to the ok state under the assumption that it is the only peer still present.

It is worth noting that the current implementation makes no effort to mitigate against the generation of packets for actions requiring "immediate" transmission. ie, if a peer requests uncompressed updates for 8 states, the current pfsync code will generate 8 separate packets, one for each of the uncompressed updates it generates.

Also, because pfsync only builds one type of packet at a time, it is susceptible to generating excessive traffic simply because pf notifies it of different events all the time. It is uncommon for a state insert to be followed immediately by another state insert. The same is true for state deletions. It is common for pfsync to be able to build packets for several updates and mitigate against frequent sends of those packets, but that mitigation is offset by the inserts and deletes that occur too.

It is also worth noting that this implementation makes no attempt to detect if the traffic associated with a particular state is split over two of the systems in the cluster. For traffic going through a pf firewall that requires up to date state information to proceed, the lack of this detection will prevent the traffic from moving forward.

For example, because pf tracks a TCP sessions sequence numbers, and because TCP uses packets in both directions to move those sequence numbers forward, a pf firewall needs frequent updates from its peers if it can only see half the TCP packets. Without an update from a peer pf will not move the states sequence numbers forward. pf will drop TCP packets that move beyond the TCP sequence number windows (plus a bit of fuzz).

Because it is almost impossible for a TCP session to receive 128 updates (ie, 128 new packets from the wire) without moving the session forward, it is in the worst case only the 1 second timeout which causes pfsync to send its update out. Therefore an active-active pf cluster may only allow new packets in a TCP session through for a small portion of every second.

In response to this the TCP endpoints will back off the speed at which they send and attempt retransmission. The TCP session will move forward, but only at a small fraction of the speed that a single pf firewall would forward at.

This over mitigation of messages proves to be the fatal flaw that prevents pfsync and pf to be usable in active-active clusters.

Currently pfsync only generates and receives IPv4 datagrams.

Approach and Execution

As described in detail above, the big problem with the current implementation is that it mitigates sending of pfsync packets too much, ie, in a firewall cluster with traffic split over two peers, updates aren't exchanged rapidly enough for the states on each firewall to move forward fast enough to keep up with the actual traffic. This is especially (or perhaps only) problematic with TCP traffic, which requires extremely current information from both sides of the connection to move the TCP sequence numbers forward.

Two attempts were made to try to solve the active-active problem in pfsync, firstly simple modifications to the existing implementation, and then as a result of that an almost full blown rewrite of the code.

Changes To The pfsync v4 Implementation

A large number of different approaches at dealing with states with traffic split over two peers in an active-active firewall cluster were evaluated and tested as part of the initial problem solving. This stage could be considered the exploratory surgery and was required so I could gain familiarity with the problem and the current implementation. However, all of the solutions except the final solution presented here were rejected as being unsuitable.

These solutions ranged from allowing packets for split TCP states to accept packets if the sequence numbers are on the edge of the window as stored in the state, to decreasing the maximum timeout on pfsync packets from 1 second down to small fractions of a second. All of these solutions either compromised the security provided by pf, or hurt the performance too much in existing use cases to be feasible.

Despite the long road to the changes made to the v4 code, the final changes were actually quite minimal. After a lot of trial and error it was decided that it was necessary for each firewall involved in a split path for TCP sessions to be notified of updates to that state as soon as possible. This in turn required the ability to detect when two peers were involved in a split path.

Detecting a split path turned out to be simple. Whenever an update to a state is received via the pfsync protocol, we record the time the update arrived at in the pf state structure. Then, when an update to the same state arrives via pf, we simply find the difference between the current time and the last time the state was updated via pf and assume the state is split if the difference is less than some arbitrary value, 4 in our case.

If we never get an update about a state via pfsync it means no other peer was involved in handling that state, therefore the timestamp in the state will always be 0 (the default value). The comparison between it and the current time will always be greater than 4. A peer that is handling packets for that session will send pfsync packets out about it though, so that comparison will evaluate to true and we know the state is split at that point.

The other feature of this mechanism is if the paths in both directions for this state merge onto a single firewall, that firewall will no longer receive updates for the state via pfsync. The timestamp on the state will no longer be updated, and the comparison between it and the system time will eventually fail as time moves forward. This means the same mechanism for detecting split states also allows us to detect when a state is no longer split.

With that mechanism in place it was trivial to modify the code to immediately send a pfsync state update about our own state changes to the peer.

These changes were successful, ie, if you had a pair of firewalls called A and B between two networks x and y, you could configure the route from hosts on network x to network y to go via firewall A, and the routes from hosts on network y to network x on firewall B, you could then successfully run pf as a fully stateful firewall with traffic for the same session split between those two firewalls.

The problem with these code changes is that they caused pfsync to become extremely chatty. Every single packet involved in a split session going through a firewall would generate a corresponding update from pfsync. Worse, for every single update for that split session we received from the firewall peers, we hit the merge state case in the pfsync update parsing code which cause us to send an update out again. Because of this the majority of the test we did with async paths for traffic through firewalls showed the pfsync traffic between two firewalls was several times the traffic of the actual traffic we were forwarding over the firewalls. Obviously causing more load than what we’re attempting to filter is unacceptable.

One of the discoveries made during the tinkering with the v4 code was that there was a race between the forwarding of a packet that caused the creation of a state, the pfsync packet it generates, and when the reply from the host the packet was for is seen by a peer.

If we forward that packet on to the host, and that host sends a reply through another firewall, it is likely that the 1 second timeout on the pfsync packet describing that state has not hit that second firewall yet. Because pf is stateful, it will probably reject or drop that reply rather than forward it on like it should.

In response to this problem a new pfsync message was created called PFSYNC_ACT_IACK. When a firewall creates a state, instead of forwarding the network packet that created the state on immediately, we delay transmission for a short period. While that first packet is delayed we immediately send a pfsync state insertion message. Peers that receive that state insertion message then send an insert acknowledgement message to the first firewall, which in turn uses that to trigger the transmission of the packet that was delayed. If no firewall is there to acknowledge the insert, a timeout on the packet fires and causes it to be transmitted anyway.

It was at about this point that it was decided that the code required significant surgery to avoid transmitting too many pfsync packets. Since the code was going to have to be heavily modified to fix it’s behaviour, slipping an update to the wire protocol was also allowed, especially if it would help mitigate the number of packets pfsync intended to transmit.

To summarise, it was determined that not only does the pfsync code mitigate sending of pfsync packets too much, it also doesn't mitigate them enough.

pfsync v5

The only really major flaw with version 4 of the pfsync protocol was its inability to contain multiple types of messages within the same frame. It could only contain packets with state inserts, or updates, or deletions, or so on, but not a mix of those message types. This becomes a problem if you're trying to mitigate the number of packets sent, but needed to send a lot of messages of different types.

Therefore the only real change between pfsync v4 and pfsync v5 was the removal of the message type and counter fields in the pfsync header, and the introduction of a sub-header. Several different messages can now be placed in a pfsync packet, all prefixed by different sub-headers.

A new message type was added to the protocol as part of the new version too. To cope with the race between forwarding a packet that creates a state, and it's reply hitting a peer before the state was received by that peer, it was envisaged that the first peer forwarding the first packet would defer sending of that packet until it received an acknowledgement of the insert from a peer. That acknowledgement is represented by a new PFSYNC_ACT_IACK message type that simply contains the creator and state ids of the state that was just inserted.

To summarise, pfsync v5 is largely the same as pfsync v4, except for a new message type (IACK) and the ability to send multiple types of messages inside a single pfsync packet due to the addition of a sub-header in the frame format.

The following changes to the OpenBSD kernel were made to address the inadequacies discovered by the previous implementation.

Firstly, the pfsync packet parsing code in the kernel has been split up to avoid the use of switch statements to pick which code is used to parse which message types. Switch statements in C have an O(n) cost where n is the number of options you're switching between. Instead, the parser is picked by using the pfsync action in the sub-header id as an index into a table of function pointers. This moves the cost of picking a parser for a message to O(1) complexity.

Next, pfsync packet generation was moved from being done "on the fly" when pf notified us of updates, to being done only when a packet was ready to be transmitted. This in turn required that the state of a pf state with respect to pfsync be stored somewhere other than the packet currently being generated.

Previously the code determined if a state was already scheduled to be sent out on the wire by iterating over the packet in memory. This is another O(n) cost where n is the number of states scheduled in the current packet.

Because we also wanted to be able to send multiple types of messages in the same packet, it is now also necessary to mark what type of message the state had associated with it. Several queues for holding the states were created inside pfsync, one for each type of message that the state could appear in on the wire. The pf state structure was extended to record which of these queues it was currently held on. Now it is an O(1) cost to determine where a state is with respect to pfsync.

When the packet is eventually scheduled for transmission, the pfsync code walks all these queues and serialised the states into messages within the packet. As it does this iteration it simply marks the pf states as not being queued within pfsync anymore.

Another side effect of moving to queues of states was that it is now easy to move states between queues in response to pf notifications or requests from other peers. For example, pf itself could schedule a compressed update for a state which would leave it on the compressed update queue and marked as such. A peer can then request an uncompressed update for it. Where the previous implementation would have sent the previous message out immediately so it could begin a new packet with the uncompressed message type, the new code now can trivially figure out that it should simply remove the state from the compressed update queue and place it on the uncompressed update queue and mark it as such.

Next, a mechanism to mitigate against having to send "immediate" packets out immediately was developed. Since there are no general high resolution timers available in the OpenBSD kernel, it was decided that a new softnet interrupt handler be created specifically to flush the pfsync message queues into a packet for transmission.

Because all the events in pfsync are generated by code that is running at softnet, ie, the pf tests for network packets on both the systems input and output queues and the processing of pfsync packets, it is possible to queue updates for all the states touched during that processing and schedule a softnet interrupt for the pfsync packet generator. Because that code is running at softnet it masks the pfsync packet generator scheduled at softnet and prevents from running until after all the current packet processing is finished.

Additionally, on systems with busy network interfaces it is typical that you process several dozen packets per call to the softnet interrupt handlers. Any updates requiring immediate transmission of a pfsync packet can bundle all those updates into a single update before the packet transmission code is run.

Finally, the features from the pfsync v4 reworking were brought over to the new pfsync v5 code. The time at which a pf state was last updated by a pfsync packet is stored in the pf state. If an update from pf for a state occurred within a second of the update from the pfsync system, it is determined that the traffic for that state is now split over two peers in the cluster and it is marked for "immediate" transmission by the softnet handler.

The IACK handling made against the pfsync v4 implementation was also brought over, however, instead of the pointer to the packet and its timeout being stored in the state, a separate queue for deferred packets was added to the pfsync code. This was done because the space required for the packet pointer and the timeout was considered excessively wasteful for every state to store when it was only to be used for an extremely short period of time. This was weighed against the extra cost in terms of CPU time of handling that separate queue, which was considered worth it for the memory savings.

Because of the changes to the wire protocol, tools outside the kernel that parsed pfsync packets must be updated to understand the new packet format. The only tool in the OpenBSD source tree that parses those packets is tcpdump. This program was updated to handle the new packet format as part of this work.

Overall the changes to the OpenBSD system resulted in a unified diff to the source tree touching 10 separate files and totalling over 4000 lines of changes.

Results

During the implementation of the new version of the pfsync protocol, several problems in the OpenBSD kernel were uncovered.

Insufficient splsoftnet() Protection In The Network Stack

The pfsync code assumed that all paths into it from pf would hold splsoftnet, which was an assumption that was necessary to guarantee that the pfsync data structures were sufficiently locked.

Testing of the code during development kept showing corrupt data structures in pfsync and in the mbuf (network packet handling structures in the OpenBSD kernel) pools. These corruption's inevitably led to panics in the OpenBSD kernel. Because of this corruption, calls to splassert (a function that checks if the current CPU interrupt mask is at least as high as the level required) were added to the entry points into the pfsync driver to check if softnet was actually held.

It was discovered that there were cases when pfsync was being called without softnet being held.

When a normal network interface is brought up, ie, configured to send and receive packets, the IPv6 protocol is also enabled on that interface and immediately generates IPv6 duplicate address detection packets. These packets are built and sent up the network stack without the spl being raised

This meant that large portions of the network stack, including pf and pfsync, were being used without appropriate protection. If the system received a packet during this time it was likely that the state in these systems would become corrupted. This was indeed the case we discovered with pfsync.

As a result of this discovery several code paths from the IPv6 stack had the necessary calls to splsoftnet() added to provide the appropriate protection required by the network stack.

This fix was developed by Ryan McBride and committed to the OpenBSD source tree. Further splasserts have also been added to other parts of the network stack to try and catch any further problems.

ip_output() Panic

After pfsync generates a packet to be transmitted, it hands it to the ip_output function for it to be sent up the stack and on to the wire. A combination of three factors caused some of these packets to generate a panic.

Firstly, pfsync generates packets with the DF (don’t fragment) flags set in the IP header. This means that the network stack should not break the packet up into multiple IP frames if it is too large to transmit.

Secondly, pfsync sends to a multicast address by default. This changes how ip_output behaves internally in several ways, but most relevant here is that packets sent to multicast addresses don’t necessarily result in a route lookup.

Lastly, due to an accounting error the pfsync code would generate network packets that were too large to be transmitted. When ip_output is asked to deal with a packet larger than the interfaces MTU that has the DF flag set, it takes that opportunity to check if the route to the destination address needs to be updated with a smaller MTU.

Because the pfsync packet was being sent to a multicast address the local variable inside ip_output holding the destinations route was not set. When ip_output tried to use that variable to update the route’s MTU it generated a system trap caused by an access to an invalid memory address.

The fix to this problem was developed by Claudio Jeker. A simple check to see if the route variable was not NULL was added before any attempt to use or modify the route was done. This fix was also committed to the OpenBSD source tree.

Functional Results

Unfortunately the time taken to implement the new protocol and its handling in the OpenBSD kernel and the debug it left little time for testing and evaluation of performance. However, despite this the initial results are much better than expected.

The new code results in less pfsync messages being exchanged between pf firewalls when compared to the old code in the exact same setups. Users with active-passive setups will gain a benefit from the new code because of this, and may also gain a small amount of CPU time back due to the relative efficiency of the new implementation.

Even better, the new code makes active-active firewalls actually work.

This code was developed with the idea that async paths through pf firewalls was a worst case situation for a cluster of firewalls, and that the code modifications to support it were simply to make that case tolerable rather than unusable like the current code. Because of this it was predicted that traffic forwarded over async paths would be at a rate noticeably slower than the same traffic going over a single firewall.

This was indeed the case with the simple modifications to the pfsync v4 code base. There was a always a significant slow down with any traffic over async paths compared to the traffic sent over a single firewall.

However, it appears that the new pfsync protocol and implementation scales a lot better. Relatively slow TCP connections, (less than 10 thousand packets per second) do not experience any slow when split across async paths. At this rate it is trivial for the pfsync traffic to keep up with the rate at which the TCP session window moves forward. As the TCP packets per second increases above that threshold, the pfsync updates begin to struggle with keeping each firewalls view of the sequence numbers in sync. As a result the TCP state matching code in pf begins to drop packets that have moved beyond what it thinks the windows should be.

This result is in line with the expectation stated above. However, compared to the behaviour of the old implementation where async traffic simply stalls, this is a massive improvement.

Experience has shown that the majority of connections through a firewall tends toward lots of relatively slow streams, rather than one massively fast stream. From that point of view it is possible that the worst case behaviour with the new code will not be noticeable in practice.

The characteristics of the new protocol are also heavily dependant on the behaviour of the hardware that it is running on. The quality of the interrupt mitigation and the choice on network cards has a heavy influence on how many packets are processed at softnet. It is unfortunate that more time was not available to gain some understanding of these interactions.

Conclusion

Summary and Conclusion

The new protocol and the rewrite of the pfsync kernel code is a success. Not only does it allow active-active firewall clusters to be built, but it also improves the performance for currently supported active-passive configurations by reducing the network load of the associated pfsync traffic.

With this code it is now possible to increase the throughput between two networks by adding firewalls, rather than having to scale the performance of a single active firewall which takes all the load. Each peer in such an active-active cluster will be able to act as an independent gateway from the point of view of the client systems, but the network administrator will still have the ability to apply policy with the pf firewall and not have to give up security for the performance gained by running multiple gateways. Effort should be spent attempting to engineer the network so both the send and receive path will travel over the one firewall, but if that engineering fails it is possible that the service will degrade rather fail.

Future Work

Despite the improvement the code is still relatively immature and is not currently considered a complete replacement for the previous pfsync implementation. Testing between firewalls based on different CPU architectures is required to ensure no endian or alignment issues exist in the code base. Previously supported features such as the IPsec security association syncing also need to be tested to ensure the new version of the protocol and implementation support those features.

There are also some hard to fix issues with the new implementation when you move from sync traffic paths to async paths. The TCP state merge code in the state update input path seems to reject valid information which can leave a peer without the most recent information required to forward already existing connections.

Working through those issues should be a relatively trivial task given the right hardware, but was impossible to do in the time available.

It is also unlikely that this code will make it to the OpenBSD 4.5 release for these same reasons, but it is almost a certainty that it will be integrated into the 4.6 release. Work with other OpenBSD developers is continuing to ensure the code is reliable enough for inclusion in the source tree.

Appendices

Source changes

This is a unified diff to the OpenBSD source tree that implements the new protocol and code to support it. This applies to a -current source tree as at the 2nd of February, 2009.

```
Index: sys/net/if_pfsync.c
===================================================================
RCS file: /cvs/src/sys/net/if_pfsync.c,v
retrieving revision 1.102
diff -u -p -r1.102 if_pfsync.c
--- sys/net/if_pfsync.c    21 Dec 2008 23:41:26 -0000 1.102
+++ sys/net/if_pfsync.c    4 Feb 2009 05:18:14 -0000
@@ -37,16 +37,17 @@
 #include <sys/timeout.h>
 #include <sys/kernel.h>
 #include <sys/sysctl.h>
+#include <sys/pool.h>

 #include <net/if.h>
 #include <net/if_types.h>
 #include <net/route.h>
 #include <net/bpf.h>
+#include <net/netisr.h>
 #include <netinet/in.h>
 #include <netinet/if_ether.h>
 #include <netinet/tcp.h>
 #include <netinet/tcp_seq.h>
-#include <sys/pool.h>

 #ifdef    INET
 #include <netinet/in_systm.h>
@@ -70,15 +71,132 @@
 #include "bpfilter.h"
 #include "pfsync.h"

-#define PFSYNC_MINMTU     \
-    (sizeof(struct pfsync_header) + sizeof(struct pf_state))
+#define PFSYNC_MINPKT ( \
+    sizeof(struct ip) + \
+    sizeof(struct pfsync_header) + \
+    sizeof(struct pfsync_subheader) + \
+    sizeof(struct pfsync_eof))

-#ifdef PFSYNCDEBUG
-#define DPRINTF(x)    do { if (pfsyncdebug) printf x ; } while (0)
-int pfsyncdebug;
-#else
-#define DPRINTF(x)
-#endif
+struct pfsync_pkt {
+    struct ip *ip;
+    struct in_addr src;
+    u_int8_t flags;
+};
```

```
+
+int pfsync_input_hmac(struct mbuf *, int);
+
+int pfsync_upd_tcp(struct pf_state *, struct pfsync_state_peer *,
+        struct pfsync_state_peer *);
+
+int pfsync_in_clr(struct pfsync_pkt *, struct mbuf *, int, int);
+int pfsync_in_ins(struct pfsync_pkt *, struct mbuf *, int, int);
+int pfsync_in_iack(struct pfsync_pkt *, struct mbuf *, int, int);
+int pfsync_in_upd(struct pfsync_pkt *, struct mbuf *, int, int);
+int pfsync_in_upd_c(struct pfsync_pkt *, struct mbuf *, int, int);
+int pfsync_in_ureq(struct pfsync_pkt *, struct mbuf *, int, int);
+int pfsync_in_del(struct pfsync_pkt *, struct mbuf *, int, int);
+int pfsync_in_del_c(struct pfsync_pkt *, struct mbuf *, int, int);
+int pfsync_in_bus(struct pfsync_pkt *, struct mbuf *, int, int);
+int pfsync_in_tdb(struct pfsync_pkt *, struct mbuf *, int, int);
+int pfsync_in_eof(struct pfsync_pkt *, struct mbuf *, int, int);
+
+int pfsync_in_error(struct pfsync_pkt *, struct mbuf *, int, int);
+
+int (*pfsync_acts[])(struct pfsync_pkt *, struct mbuf *, int, int) = {
+    pfsync_in_clr,            /* PFSYNC_ACT_CLR */
+    pfsync_in_ins,            /* PFSYNC_ACT_INS */
+    pfsync_in_iack,           /* PFSYNC_ACT_INS_ACK */
+    pfsync_in_upd,            /* PFSYNC_ACT_UPD */
+    pfsync_in_upd_c,     /* PFSYNC_ACT_UPD_C */
+    pfsync_in_ureq,           /* PFSYNC_ACT_UPD_REQ */
+    pfsync_in_del,            /* PFSYNC_ACT_DEL */
+    pfsync_in_del_c,     /* PFSYNC_ACT_DEL_C */
+    pfsync_in_error,     /* PFSYNC_ACT_INS_F */
+    pfsync_in_error,     /* PFSYNC_ACT_DEL_F */
+    pfsync_in_bus,            /* PFSYNC_ACT_BUS */
+    pfsync_in_tdb,            /* PFSYNC_ACT_TDB */
+    pfsync_in_eof             /* PFSYNC_ACT_EOF */
+};
+
+struct pfsync_q {
+    int        (*write)(struct pf_state *, struct mbuf *, int);
+    size_t          len;
+    u_int8_t   action;
+};
+
+/* we have one of these for every PFSYNC_S_ */
+int pfsync_out_state(struct pf_state *, struct mbuf *, int);
+int pfsync_out_iack(struct pf_state *, struct mbuf *, int);
+int pfsync_out_upd_c(struct pf_state *, struct mbuf *, int);
+int pfsync_out_del(struct pf_state *, struct mbuf *, int);
+
+struct pfsync_q pfsync_qs[] = {
+    { pfsync_out_state, sizeof(struct pfsync_state),
PFSYNC_ACT_INS },
+    { pfsync_out_iack,  sizeof(struct pfsync_ins_ack),
PFSYNC_ACT_INS_ACK },
+    { pfsync_out_state, sizeof(struct pfsync_state),
PFSYNC_ACT_UPD },
+	{ pfsync_out_upd_c, sizeof(struct pfsync_upd_c),
PFSYNC_ACT_UPD_C },
+	{ pfsync_out_del,   sizeof(struct pfsync_del_c),
PFSYNC_ACT_DEL_C }
+};
+
+void	pfsync_q_ins(struct pf_state *, int);
+void	pfsync_q_del(struct pf_state *);
+
+struct pfsync_upd_req_item {
+	TAILQ_ENTRY(pfsync_upd_req_item)	ur_entry;
+	struct pfsync_upd_req			ur_msg;
+};
+TAILQ_HEAD(pfsync_upd_reqs, pfsync_upd_req_item);
+
+struct pfsync_deferral {
+	TAILQ_ENTRY(pfsync_deferral)		 pd_entry;
+	struct pf_state				*pd_st;
+	struct mbuf				*pd_m;
+	struct timeout				 pd_tmo;
+};
+TAILQ_HEAD(pfsync_deferrals, pfsync_deferral);
+
+#define PFSYNC_PLSIZE	MAX(sizeof(struct pfsync_upd_req_item), \
+			    sizeof(struct pfsync_deferral))
+
+int	pfsync_out_tdb(struct tdb *, struct mbuf *, int);
+
+struct pfsync_softc {
+	struct ifnet		 sc_if;
+	struct ifnet		*sc_sync_if;
+
+	struct pool		 sc_pool;
+
+	struct ip_moptions	 sc_imo;
+
+	struct in_addr		 sc_sync_peer;
+	u_int8_t		 sc_maxupdates;
+
+	struct ip		 sc_template;
+
+	struct pf_state_queue	 sc_qs[PFSYNC_S_COUNT];
+	size_t			 sc_len;
+
+	struct pfsync_upd_reqs	 sc_upd_req_list;
+
+	struct pfsync_deferrals	 sc_deferrals;
+	u_int			 sc_deferred;
+
+	void			*sc_plus;
+	size_t			 sc_pluslen;
+
+	u_int32_t		 sc_ureq_sent;
+	int			 sc_bulk_tries;
+	struct timeout		 sc_bulkfail_tmo;
+
+	u_int32_t		sc_ureq_received;
+	struct pf_state		*sc_bulk_next;
+	struct pf_state		*sc_bulk_last;
+	struct timeout		 sc_bulk_tmo;
+
+	TAILQ_HEAD(, tdb)	 sc_tdb_q;
+
+	struct timeout		 sc_tmo;
+};

 struct pfsync_softc	*pfsyncif = NULL;
 struct pfsyncstats	 pfsyncstats;
@@ -86,7 +204,6 @@ struct pfsyncstats	pfsyncstats;
 void	pfsyncattach(int);
 int	pfsync_clone_create(struct if_clone *, int);
 int	pfsync_clone_destroy(struct ifnet *);
-void	pfsync_setmtu(struct pfsync_softc *, int);
 int	pfsync_alloc_scrub_memory(struct pfsync_state_peer *,
	    struct pf_state_peer *);
 void	pfsync_update_net_tdb(struct pfsync_tdb *);
@@ -95,51 +212,31 @@ int	pfsyncoutput(struct ifnet *, struct
 int	pfsyncioctl(struct ifnet *, u_long, caddr_t);
 void	pfsyncstart(struct ifnet *);

-struct mbuf *pfsync_get_mbuf(struct pfsync_softc *, u_int8_t, void **);
-int	pfsync_request_update(struct pfsync_state_upd *, struct in_addr *);
-int	pfsync_sendout(struct pfsync_softc *);
+struct mbuf *pfsync_if_dequeue(struct ifnet *);
+struct mbuf *pfsync_get_mbuf(struct pfsync_softc *);
+
+void	pfsync_deferred(struct pf_state *, int);
+void	pfsync_undefer(struct pfsync_deferral *, int);
+void	pfsync_defer_tmo(void *);
+
+void	pfsync_request_update(u_int32_t, u_int64_t);
+void	pfsync_update_state_req(struct pf_state *);
+
+void	pfsync_drop(struct pfsync_softc *);
+void	pfsync_sendout(void);
+void	pfsync_send_plus(void *, size_t);
 int	pfsync_tdb_sendout(struct pfsync_softc *);
 int	pfsync_sendout_mbuf(struct pfsync_softc *, struct mbuf *);
 void	pfsync_timeout(void *);
 void	pfsync_tdb_timeout(void *);
 void	pfsync_send_bus(struct pfsync_softc *, u_int8_t);
-void	pfsync_bulk_update(void *);
-void	pfsync_bulkfail(void *);
-
-struct pfsync_pkt {
-	struct ip *ip;
-	struct in_addr src;
-	u_int8_t flags;
-};
-
-void	pfsync4_input(struct pfsync_pkt *, struct mbuf *);
```

```
-int pfsync4_in_clr(struct pfsync_pkt *, struct mbuf *, int, int);
-int pfsync4_in_ins(struct pfsync_pkt *, struct mbuf *, int, int);
-int pfsync4_in_upd(struct pfsync_pkt *, struct mbuf *, int, int);
-int pfsync4_in_del(struct pfsync_pkt *, struct mbuf *, int, int);
-int pfsync4_in_upd_c(struct pfsync_pkt *, struct mbuf *, int, int);
-int pfsync4_in_del_c(struct pfsync_pkt *, struct mbuf *, int, int);
-int pfsync4_in_ureq(struct pfsync_pkt *, struct mbuf *, int, int);
-int pfsync4_in_bus(struct pfsync_pkt *, struct mbuf *, int, int);
-int pfsync4_in_tdb_upd(struct pfsync_pkt *, struct mbuf *, int, int);
-
-int pfsync4_in_error(struct pfsync_pkt *, struct mbuf *, int, int);
-
-int (*pfsync4_acts[])(struct pfsync_pkt *, struct mbuf *, int, int) = {
-    pfsync4_in_clr,           /* PFSYNC_ACT_CLR */
-    pfsync4_in_ins,           /* PFSYNC_ACT_INS */
-    pfsync4_in_upd,           /* PFSYNC_ACT_UPD */
-    pfsync4_in_del,           /* PFSYNC_ACT_DEL */
-    pfsync4_in_upd_c,         /* PFSYNC_ACT_UPD_C */
-    pfsync4_in_del_c,         /* PFSYNC_ACT_DEL_C */
-    pfsync4_in_error,         /* PFSYNC_ACT_INS_F */
-    pfsync4_in_error,         /* PFSYNC_ACT_DEL_F */
-    pfsync4_in_ureq,      /* PFSYNC_ACT_UREQ */
-    pfsync4_in_bus,           /* PFSYNC_ACT_BUS */
-    pfsync4_in_tdb_upd        /* PFSYNC_ACT_TDB_UPD */
-};
+void pfsync_bulk_start(void);
+void pfsync_bulk_status(u_int8_t);
+void pfsync_bulk_update(void *);
+void pfsync_bulk_fail(void *);

+#define PFSYNC_MAX_BULKTRIES   12
 int pfsync_sync_ok;

 struct if_clonepfsync_cloner =
@@ -153,46 +250,50 @@ pfsyncattach(int npfsync)
 int
 pfsync_clone_create(struct if_clone *ifc, int unit)
 {
+    struct pfsync_softc *sc;
     struct ifnet *ifp;
+    int q;

     if (unit != 0)
          return (EINVAL);

     pfsync_sync_ok = 1;
-    if ((pfsyncif = malloc(sizeof(*pfsyncif), M_DEVBUF,
-        M_NOWAIT|M_ZERO)) == NULL)
+
+    sc = malloc(sizeof(*pfsyncif), M_DEVBUF, M_NOWAIT | M_ZERO);
+    if (sc == NULL)
          return (ENOMEM);
-    pfsyncif->sc_mbuf = NULL;
-    pfsyncif->sc_mbuf_net = NULL;
-    pfsyncif->sc_mbuf_tdb = NULL;
-    pfsyncif->sc_statep.s = NULL;
```

```
-	pfsyncif->sc_statep_net.s = NULL;
-	pfsyncif->sc_statep_tdb.t = NULL;
-	pfsyncif->sc_maxupdates = 128;
-	pfsyncif->sc_sync_peer.s_addr = INADDR_PFSYNC_GROUP;
-	pfsyncif->sc_sendaddr.s_addr = INADDR_PFSYNC_GROUP;
-	pfsyncif->sc_ureq_received = 0;
-	pfsyncif->sc_ureq_sent = 0;
-	pfsyncif->sc_bulk_send_next = NULL;
-	pfsyncif->sc_bulk_terminator = NULL;
-	pfsyncif->sc_imo.imo_membership = (struct in_multi **)malloc(
+
+	for (q = 0; q < PFSYNC_S_COUNT; q++)
+		TAILQ_INIT(&sc->sc_qs[q]);
+
+	pool_init(&sc->sc_pool, PFSYNC_PLSIZE, 0, 0, 0, "pfsync", NULL);
+	TAILQ_INIT(&sc->sc_upd_req_list);
+	TAILQ_INIT(&sc->sc_deferrals);
+	sc->sc_deferred = 0;
+
+	sc->sc_len = PFSYNC_MINPKT;
+	sc->sc_maxupdates = 128;
+
+	sc->sc_imo.imo_membership = (struct in_multi **)malloc(
	    (sizeof(struct in_multi *) * IP_MIN_MEMBERSHIPS), M_IPMOPTS,
-	    M_WAITOK|M_ZERO);
-	pfsyncif->sc_imo.imo_max_memberships = IP_MIN_MEMBERSHIPS;
-	ifp = &pfsyncif->sc_if;
+	    M_WAITOK | M_ZERO);
+	sc->sc_imo.imo_max_memberships = IP_MIN_MEMBERSHIPS;
+
+	ifp = &sc->sc_if;
	snprintf(ifp->if_xname, sizeof ifp->if_xname, "pfsync%d", unit);
-	ifp->if_softc = pfsyncif;
+	ifp->if_softc = sc;
	ifp->if_ioctl = pfsyncioctl;
	ifp->if_output = pfsyncoutput;
	ifp->if_start = pfsyncstart;
	ifp->if_type = IFT_PFSYNC;
	ifp->if_snd.ifq_maxlen = ifqmaxlen;
-	ifp->if_hdrlen = PFSYNC_HDRLEN;
-	pfsync_setmtu(pfsyncif, ETHERMTU);
-	timeout_set(&pfsyncif->sc_tmo, pfsync_timeout, pfsyncif);
-	timeout_set(&pfsyncif->sc_tdb_tmo, pfsync_tdb_timeout, pfsyncif);
-	timeout_set(&pfsyncif->sc_bulk_tmo, pfsync_bulk_update, pfsyncif);
-	timeout_set(&pfsyncif->sc_bulkfail_tmo, pfsync_bulkfail, pfsyncif);
+	ifp->if_hdrlen = sizeof(struct pfsync_header);
+	ifp->if_mtu = 1500; /* XXX */
+	ifp->if_hardmtu = MCLBYTES; /* XXX */
+	timeout_set(&sc->sc_tmo, pfsync_timeout, sc);
+	timeout_set(&sc->sc_bulk_tmo, pfsync_bulk_update, sc);
+	timeout_set(&sc->sc_bulkfail_tmo, pfsync_bulk_fail, sc);
+
	if_attach(ifp);
	if_alloc_sadl(ifp);

@@ -201,9 +302,11 @@ pfsync_clone_create(struct if_clone *ifc
 #endif

 #if NBPFILTER > 0
-	bpfattach(&pfsyncif->sc_if.if_bpf, ifp, DLT_PFSYNC, PFSYNC_HDRLEN);
+	bpfattach(&sc->sc_if.if_bpf, ifp, DLT_PFSYNC, PFSYNC_HDRLEN);
 #endif

+	pfsyncif = sc;
+
	return (0);
 }

@@ -212,10 +315,8 @@ pfsync_clone_destroy(struct ifnet *ifp)
 {
	struct pfsync_softc *sc = ifp->if_softc;

-	timeout_del(&sc->sc_tmo);
-	timeout_del(&sc->sc_tdb_tmo);
	timeout_del(&sc->sc_bulk_tmo);
-	timeout_del(&sc->sc_bulkfail_tmo);
+	timeout_del(&sc->sc_tmo);
 #if NCARP > 0
	if (!pfsync_sync_ok)
		carp_group_demote_adj(&sc->sc_if, -1);
@@ -224,12 +325,34 @@ pfsync_clone_destroy(struct ifnet *ifp)
	bpfdetach(ifp);
 #endif
	if_detach(ifp);
-	free(pfsyncif->sc_imo.imo_membership, M_IPMOPTS);
-	free(pfsyncif, M_DEVBUF);
+
+	pfsync_drop(sc);
+
+	while (sc->sc_deferred > 0)
+		pfsync_undefer(TAILQ_FIRST(&sc->sc_deferrals), 0);
+
+	pool_destroy(&sc->sc_pool);
+	free(sc->sc_imo.imo_membership, M_IPMOPTS);
+	free(sc, M_DEVBUF);
+
	pfsyncif = NULL;
+
	return (0);
 }

+struct mbuf *
+pfsync_if_dequeue(struct ifnet *ifp)
+{
+	struct mbuf *m;
+	int s;
+
+	s = splnet();
+	IF_DEQUEUE(&ifp->if_snd, m);
+	splx(s);
+
+	return (m);
```

```
+}
+
 /*
  * Start output on the pfsync interface.
  */
@@ -237,18 +360,10 @@ void
 pfsyncstart(struct ifnet *ifp)
 {
 	struct mbuf *m;
-	int s;
 
-	for (;;) {
-		s = splnet();
+	while ((m = pfsync_if_dequeue(ifp)) != NULL) {
 		IF_DROP(&ifp->if_snd);
-		IF_DEQUEUE(&ifp->if_snd, m);
-		splx(s);
-
-		if (m == NULL)
-			return;
-		else
-			m_freem(m);
+		m_freem(m);
 	}
 }
 
@@ -423,8 +538,6 @@ pfsync_state_import(struct pfsync_state
 	st->log = sp->log;
 	st->timeout = sp->timeout;
 	st->state_flags = sp->state_flags;
-	if (!(flags & PFSYNC_SI_IOCTL))
-		st->sync_flags = PFSTATE_FROMSYNC;
 
 	bcopy(sp->id, &st->id, sizeof(st->id));
 	st->creatorid = sp->creatorid;
@@ -436,19 +549,31 @@ pfsync_state_import(struct pfsync_state
 	st->anchor.ptr = NULL;
 	st->rt_kif = NULL;
 
-	st->pfsync_time = 0;
-
+	st->pfsync_time = time_second;
+	st->sync_state = PFSYNC_S_NONE;
 
 	/* XXX when we have nat_rule/anchors, use STATE_INC_COUNTERS */
 	r->states_cur++;
 	r->states_tot++;
 
+	if (!ISSET(flags, PFSYNC_SI_IOCTL))
+		SET(st->state_flags, PFSTATE_NOSYNC);
+
 	if ((error = pf_state_insert(kif, skw, sks, st)) != 0) {
 		/* XXX when we have nat_rule/anchors, use STATE_DEC_COUNTERS
*/
 		r->states_cur--;
 		goto cleanup_state;
```

```
     }

+    if (!ISSET(flags, PFSYNC_SI_IOCTL)) {
+         CLR(st->state_flags, PFSTATE_NOSYNC);
+         if (ISSET(st->state_flags, PFSTATE_ACK)) {
+              pfsync_q_ins(st, PFSYNC_S_IACK);
+              schednetisr(NETISR_PFSYNC);
+         }
+    }
+    CLR(st->state_flags, PFSTATE_ACK);
+
     return (0);

  cleanup:
@@ -478,31 +603,33 @@ pfsync_input(struct mbuf *m, ...)
     struct pfsync_pkt pkt;
     struct ip *ip = mtod(m, struct ip *);
     struct pfsync_header *ph;
+    struct pfsync_subheader subh;

     int offset;
-    int action, count;
     int rv;

     pfsyncstats.pfsyncs_ipackets++;

     /* verify that we have a sync interface configured */
-    if (!sc || !sc->sc_sync_ifp || !pf_status.running)
+    if (!sc || !sc->sc_sync_if || !pf_status.running)
          goto done;

     /* verify that the packet came in on the right interface */
-    if (sc->sc_sync_ifp != m->m_pkthdr.rcvif) {
+    if (sc->sc_sync_if != m->m_pkthdr.rcvif) {
          pfsyncstats.pfsyncs_badif++;
          goto done;
     }

-    /* verify that the IP TTL is 255.  */
+    sc->sc_if.if_ipackets++;
+    sc->sc_if.if_ibytes += m->m_pkthdr.len;
+
+    /* verify that the IP TTL is 255. */
     if (ip->ip_ttl != PFSYNC_DFLTTL) {
          pfsyncstats.pfsyncs_badttl++;
          goto done;
     }

     offset = ip->ip_hl << 2;
-
     if (m->m_pkthdr.len < offset + sizeof(*ph)) {
          pfsyncstats.pfsyncs_hdrops++;
          goto done;
@@ -523,103 +650,121 @@ pfsync_input(struct mbuf *m, ...)
          goto done;
     }
```

```
-	action = ph->action;
-	count = ph->count;
-
-	/* make sure it's a valid action code */
-	if (action >= PFSYNC_ACT_MAX) {
-		pfsyncstats.pfsyncs_badact++;
+#if 0
+	if (pfsync_input_hmac(m, offset) != 0) {
+		/* XXX stats */
 		goto done;
 	}
+#endif

 	/* Cheaper to grab this now than having to mess with mbufs later */
 	pkt.ip = ip;
 	pkt.src = ip->ip_src;
 	pkt.flags = 0;

-	if (!bcmp(&ph->pf_chksum, &pf_status.pf_chksum,
PF_MD5_DIGEST_LENGTH))
+	if (!bcmp(&ph->pfcksum, &pf_status.pf_chksum,
PF_MD5_DIGEST_LENGTH))
 		pkt.flags |= PFSYNC_SI_CKSUM;

 	offset += sizeof(*ph);
-	rv = (*pfsync4_acts[action])(&pkt, m, offset, count);
-	if (rv == -1)
-		return;
+	for (;;) {
+		m_copydata(m, offset, sizeof(subh), (caddr_t)&subh);
+		offset += sizeof(subh);
+
+		if (subh.action >= PFSYNC_ACT_MAX) {
+			pfsyncstats.pfsyncs_badact++;
+			goto done;
+		}
+
+		rv = (*pfsync_acts[subh.action])(&pkt, m, offset,
+		    ntohs(subh.count));
+		if (rv == -1)
+			return;
+
+		offset += rv;
+	}

 done:
 	m_freem(m);
 }

 int
-pfsync4_in_clr(struct pfsync_pkt *pkt, struct mbuf *m, int offset, int
count)
+pfsync_in_clr(struct pfsync_pkt *pkt, struct mbuf *m, int offset, int
count)
 {
```

```
+       struct pfsync_clr *clr;
        struct mbuf *mp;
-       int offp;
+       int len = sizeof(*clr) * count;
+       int i, offp;

-       struct pfsync_state_clr *cp;
-       struct pf_state *nexts;
+       struct pf_state *st, *nexts;
        struct pf_state_key *sk, *nextsk;
-           struct pf_state_item *si;
-       struct pf_state *st;
-       struct pfi_kif *kif;
+       struct pf_state_item *si;
        u_int32_t creatorid;
        int s;

-       mp = m_pulldown(m, offset, sizeof(*cp), &offp);
+       mp = m_pulldown(m, offset, len, &offp);
        if (mp == NULL) {
                pfsyncstats.pfsyncs_badlen++;
                return (-1);
        }
-       cp = (struct pfsync_state_clr *)(mp->m_data + offp);
+       clr = (struct pfsync_clr *)(mp->m_data + offp);

        s = splsoftnet();
-       if (cp->ifname[0] == '\0') {
-               for (st = RB_MIN(pf_state_tree_id, &tree_id); st; st = nexts)
{
-                       nexts = RB_NEXT(pf_state_tree_id, &tree_id, st);
-                       if (st->creatorid == creatorid) {
-                               st->sync_flags |= PFSTATE_FROMSYNC;
-                               pf_unlink_state(st);
+       for (i = 0; i < count; i++) {
+               creatorid = clr[i].creatorid;
+
+               if (clr[i].ifname[0] == '\0') {
+                       for (st = RB_MIN(pf_state_tree_id, &tree_id);
+                           st; st = nexts) {
+                               nexts = RB_NEXT(pf_state_tree_id, &tree_id, st);
+                               if (st->creatorid == creatorid) {
+                                       SET(st->state_flags, PFSTATE_NOSYNC);
+                                       pf_unlink_state(st);
+                               }
                        }
-               }
-       } else if ((kif = pfi_kif_get(cp->ifname)) != NULL) {
-               /* XXX correct? */
-               for (sk = RB_MIN(pf_state_tree, &pf_statetbl);
-                   sk; sk = nextsk) {
-                       nextsk = RB_NEXT(pf_state_tree, &pf_statetbl, sk);
-                       TAILQ_FOREACH(si, &sk->states, entry) {
-                               if (si->s->creatorid == creatorid &&
-                                   si->s->kif == kif) {
-                                       si->s->sync_flags |= PFSTATE_FROMSYNC;
-                         pf_unlink_state(si->s);
+           } else {
+                if (pfi_kif_get(clr[i].ifname) == NULL)
+                     continue;
+
+                /* XXX correct? */
+                for (sk = RB_MIN(pf_state_tree, &pf_statetbl);
+                    sk; sk = nextsk) {
+                     nextsk = RB_NEXT(pf_state_tree,
+                         &pf_statetbl, sk);
+                     TAILQ_FOREACH(si, &sk->states, entry) {
+                          if (si->s->creatorid == creatorid) {
+                               SET(si->s->state_flags,
+                                   PFSTATE_NOSYNC);
+                               pf_unlink_state(si->s);
+                          }
                      }
                 }
            }
      }
      splx(s);

-     return (sizeof(*cp));
+     return (len);
 }

 int
-pfsync4_in_ins(struct pfsync_pkt *pkt, struct mbuf *m, int offset, int
count)
+pfsync_in_ins(struct pfsync_pkt *pkt, struct mbuf *m, int offset, int
count)
 {
-     struct pfsync_state *sp, *spa;
-     int s;
-
      struct mbuf *mp;
-     int len = count * sizeof(*sp);
-     int offp;
-     int i;
+     struct pfsync_state *sa, *sp;
+     int len = sizeof(*sp) * count;
+     int i, offp;
+
+     int s;

      mp = m_pulldown(m, offset, len, &offp);
      if (mp == NULL) {
           pfsyncstats.pfsyncs_badlen++;
           return (-1);
      }
-     spa = (struct pfsync_state *)(mp->m_data + offp);
+     sa = (struct pfsync_state *)(mp->m_data + offp);

      s = splsoftnet();
      for (i = 0; i < count; i++) {
-          sp = &spa[i];
```

```
+          sp = &sa[i];

           /* check for invalid values */
           if (sp->timeout >= PFTM_MAX ||
@@ -628,7 +773,7 @@ pfsync4_in_ins(struct pfsync_pkt *pkt, s
               sp->direction > PF_OUT ||
               (sp->af != AF_INET && sp->af != AF_INET6)) {
                if (pf_status.debug >= PF_DEBUG_MISC) {
-                    printf("pfsync_input: PFSYNC_ACT_INS: "
+                    printf("pfsync_input: PFSYNC5_ACT_INS: "
                         "invalid value\n");
                }
                pfsyncstats.pfsyncs_badval++;
@@ -636,7 +781,7 @@ pfsync4_in_ins(struct pfsync_pkt *pkt, s
           }

           if (pfsync_state_import(sp, pkt->flags) == ENOMEM) {
-               /* drop out */
+               /* drop out, but process the rest of the actions */
                break;
           }
      }
@@ -646,34 +791,129 @@ pfsync4_in_ins(struct pfsync_pkt *pkt, s
 }

 int
-pfsync4_in_upd(struct pfsync_pkt *pkt, struct mbuf *m, int offset, int
count)
+pfsync_in_iack(struct pfsync_pkt *pkt, struct mbuf *m, int offset, int
count)
 {
-     struct pfsync_softc *sc = pfsyncif;
-     struct pfsync_state *sp, *spa;
+     struct pfsync_ins_ack *ia, *iaa;
      struct pf_state_cmp id_key;
-     struct pf_state_key *sk;
      struct pf_state *st;
-     int stale, sfail;
-     int flags;
+
+     struct mbuf *mp;
+     int len = count * sizeof(*ia);
+     int offp, i;
      int s;

+     mp = m_pulldown(m, offset, len, &offp);
+     if (mp == NULL) {
+          pfsyncstats.pfsyncs_badlen++;
+          return (-1);
+     }
+     iaa = (struct pfsync_ins_ack *)(mp->m_data + offp);
+
+     s = splsoftnet();
+     for (i = 0; i < count; i++) {
+          ia = &iaa[i];
+
+		bcopy(&ia->id, &id_key.id, sizeof(id_key.id));
+		id_key.creatorid = ia->creatorid;
+
+		st = pf_find_state_byid(&id_key);
+		if (st == NULL)
+			continue;
+
+		if (ISSET(st->state_flags, PFSTATE_ACK))
+			pfsync_deferred(st, 0);
+	}
+	splx(s);
+	/*
+	 * XXX this is not yet implemented, but we know the size of the
+	 * message so we can skip it.
+	 */
+
+	return (count * sizeof(struct pfsync_ins_ack));
+}
+
+int
+pfsync_upd_tcp(struct pf_state *st, struct pfsync_state_peer *src,
+    struct pfsync_state_peer *dst)
+{
+	int sfail = 0;
+	int split = 0;
+
+	if (st->src.state < src->state ||
+	    !SEQ_GT(st->src.seqlo, ntohl(src->seqlo))) {
+		pf_state_peer_ntoh(src, &st->src);
+		split++;
+	}
+	if (st->dst.state < dst->state ||
+	    !SEQ_GT(st->dst.seqlo, ntohl(dst->seqlo))) {
+		pf_state_peer_ntoh(dst, &st->dst);
+		split++;
+	}
+
+	switch (split) {
+	case 0:
+	case 1:
+		sfail = 1;
+		break;
+	case 2:
+		sfail = -1;
+		break;
+	}
+#if 0
+	/*
+	 * The state should never go backwards except
+	 * for syn-proxy states.  Neither should the
+	 * sequence window slide backwards.
+	 */
+	if (st->src.state > src->state &&
+	    (st->src.state < PF_TCPS_PROXY_SRC ||
+	    src->state >= PF_TCPS_PROXY_SRC))
+		sfail = 1;
```

```
+	else if (SEQ_GT(st->src.seqlo, ntohl(src->seqlo)))
+		sfail = 3;
+	else if (st->dst.state > dst->state) {
+		/* There might still be useful
+		 * information about the src state here,
+		 * so import that part of the update,
+		 * then "fail" so we send the updated
+		 * state back to the peer who is missing
+		 * our what we know. */
+		pf_state_peer_ntoh(src, &st->src);
+		/* XXX do anything with timeouts? */
+		sfail = 7;
+	} else if (st->dst.state >= TCPS_SYN_SENT &&
+	    SEQ_GT(st->dst.seqlo, ntohl(dst->seqlo)))
+		sfail = 4;
+#endif
+
+	return (sfail);
+}
+
+int
+pfsync_in_upd(struct pfsync_pkt *pkt, struct mbuf *m, int offset, int
count)
+{
+//	struct pfsync_softc *sc = pfsyncif;
+	struct pfsync_state *sa, *sp;
+	struct pf_state_cmp id_key;
+	struct pf_state_key *sk;
+	struct pf_state *st;
+	int sfail;
+
 	struct mbuf *mp;
 	int len = count * sizeof(*sp);
-	int offp;
-	int i;
+	int offp, i;
+	int s;

 	mp = m_pulldown(m, offset, len, &offp);
 	if (mp == NULL) {
 		pfsyncstats.pfsyncs_badlen++;
 		return (-1);
 	}
-	spa = (struct pfsync_state *)(mp->m_data + offp);
+	sa = (struct pfsync_state *)(mp->m_data + offp);

 	s = splsoftnet();
 	for (i = 0; i < count; i++) {
-		sp = &spa[i];
-
-		flags = PFSYNC_FLAG_STALE;
+		sp = &sa[i];

 		/* check for invalid values */
 		if (sp->timeout >= PFTM_MAX ||
@@ -693,40 +933,19 @@ pfsync4_in_upd(struct pfsync_pkt *pkt, s
		st = pf_find_state_byid(&id_key);
		if (st == NULL) {
			/* insert the update */
-			if (pfsync_state_import(sp, flags))
+			if (pfsync_state_import(sp, 0))
				pfsyncstats.pfsyncs_badstate++;
			continue;
		}
-		sk = st->key[PF_SK_WIRE];  /* XXX right one? */

+		if (ISSET(st->state_flags, PFSTATE_ACK))
+			pfsync_deferred(st, 1);
+
+		sk = st->key[PF_SK_WIRE];  /* XXX right one? */
		sfail = 0;
-		if (sk->proto == IPPROTO_TCP) {
-			/*
-			 * The state should never go backwards except
-			 * for syn-proxy states.  Neither should the
-			 * sequence window slide backwards.
-			 */
-			if (st->src.state > sp->src.state &&
-			    (st->src.state < PF_TCPS_PROXY_SRC ||
-			    sp->src.state >= PF_TCPS_PROXY_SRC))
-				sfail = 1;
-			else if (SEQ_GT(st->src.seqlo, ntohl(sp->src.seqlo)))
-				sfail = 3;
-			else if (st->dst.state > sp->dst.state) {
-				/* There might still be useful
-				 * information about the src state here,
-				 * so import that part of the update,
-				 * then "fail" so we send the updated
-				 * state back to the peer who is missing
-				 * our what we know. */
-				pf_state_peer_ntoh(&sp->src, &st->src);
-				/* XXX do anything with timeouts? */
-				sfail = 7;
-				flags = 0;
-			} else if (st->dst.state >= TCPS_SYN_SENT &&
-			    SEQ_GT(st->dst.seqlo, ntohl(sp->dst.seqlo)))
-				sfail = 4;
-		} else {
+		if (sk->proto == IPPROTO_TCP)
+			sfail = pfsync_upd_tcp(st, &sp->src, &sp->dst);
+		else {
			/*
			 * Non-TCP protocol state machine always go
			 * forwards
@@ -736,339 +955,279 @@ pfsync4_in_upd(struct pfsync_pkt *pkt, s
			else if (st->dst.state > sp->dst.state)
				sfail = 6;
		}
-		if (sfail) {
-			if (pf_status.debug >= PF_DEBUG_MISC)
-				printf("pfsync: %s stale update "
-				    "(%d) id: %016llx "
-                       "creatorid: %08x\n",
-                       (sfail < 7 ?  "ignoring"
-                        : "partial"), sfail,
-                       betoh64(st->id),
+
+          if (sfail > 0) {
+               if (pf_status.debug >= PF_DEBUG_MISC) {
+                    printf("pfsync: %s stale update (%d)"
+                        " id: %016llx creatorid: %08x\n",
+                        (sfail < 7 ?  "ignoring" : "partial"),
+                        sfail, betoh64(st->id),
                        ntohl(st->creatorid));
+               }
                pfsyncstats.pfsyncs_stale++;

-               if (!(sp->sync_flags & PFSTATE_STALE)) {
-                    /* we have a better state, send it */
-                    if (sc->sc_mbuf != NULL && !stale)
-                         pfsync_sendout(sc);
-                    stale++;
-                    if (!st->sync_flags)
-                         pfsync_pack_state( PFSYNC_ACT_UPD,
-                              st, flags);
-               }
+               pfsync_update_state(st);
+               schednetisr(NETISR_PFSYNC);
                continue;
+          } else if (sfail == 0) {
+               pfsync_alloc_scrub_memory(&sp->dst, &st->dst);
+               pf_state_peer_ntoh(&sp->src, &st->src);
+               pf_state_peer_ntoh(&sp->dst, &st->dst);
           }
-          pfsync_alloc_scrub_memory(&sp->dst, &st->dst);
-          pf_state_peer_ntoh(&sp->src, &st->src);
-          pf_state_peer_ntoh(&sp->dst, &st->dst);
           st->expire = ntohl(sp->expire) + time_second;
           st->timeout = sp->timeout;
+          st->pfsync_time = time_second;
      }
-     if (stale && sc->sc_mbuf != NULL)
-          pfsync_sendout(sc);
      splx(s);

      return (len);
 }

 int
-pfsync4_in_del(struct pfsync_pkt *pkt, struct mbuf *m, int offset, int
count)
+pfsync_in_upd_c(struct pfsync_pkt *pkt, struct mbuf *m, int offset, int
count)
 {
-     struct pfsync_state *sp, *spa;
+//   struct pfsync_softc *sc = pfsyncif;
+     struct pfsync_upd_c *ua, *up;
+     struct pf_state_key *sk;
```

```
 	struct pf_state_cmp id_key;
 	struct pf_state *st;
-	int s;
-
-	struct mbuf *mp;
-	int len = count * sizeof(*sp);
-	int offp;
-	int i;
-
-	mp = m_pulldown(m, offset, len, &offp);
-	if (mp == NULL) {
-		pfsyncstats.pfsyncs_badlen++;
-		return (-1);
-	}
-	spa = (struct pfsync_state *)(mp->m_data + offp);
-
-	s = splsoftnet();
-	for (i = 0; i < count; i++) {
-		sp = &spa[i];
-
-		bcopy(sp->id, &id_key.id, sizeof(id_key.id));
-		id_key.creatorid = sp->creatorid;
-
-		st = pf_find_state_byid(&id_key);
-		if (st == NULL) {
-			pfsyncstats.pfsyncs_badstate++;
-			continue;
-		}
-		st->sync_flags |= PFSTATE_FROMSYNC;
-		pf_unlink_state(st);
-	}
-	splx(s);
-
-	return (len);
-}

-int
-pfsync4_in_upd_c(struct pfsync_pkt *pkt, struct mbuf *m, int offset,
int count)
-{
-	struct pfsync_softc *sc = pfsyncif;
-	struct pfsync_state_upd *up, *upa;
-	struct pf_state_cmp id_key;
-	struct pf_state_key *sk;
-	struct pf_state *st;
-	int stale, sfail;
-	int update_requested;
-	int s;
+	int len = count * sizeof(*up);
+	int sfail;

 	struct mbuf *mp;
-	int len = count * sizeof(*up);
-	int offp;
-	int i;
+	int offp, i;
```

```
+       int s;

        mp = m_pulldown(m, offset, len, &offp);
        if (mp == NULL) {
             pfsyncstats.pfsyncs_badlen++;
             return (-1);
        }
-       upa = (struct pfsync_state_upd *)(mp->m_data + offp);
+       ua = (struct pfsync_upd_c *)(mp->m_data + offp);

-       s = splnet();
+       s = splsoftnet();
        for (i = 0; i < count; i++) {
-            up = &upa[i];
+            up = &ua[i];

             /* check for invalid values */
             if (up->timeout >= PFTM_MAX ||
                 up->src.state > PF_TCPS_PROXY_DST ||
                 up->dst.state > PF_TCPS_PROXY_DST) {
-                  if (pf_status.debug >= PF_DEBUG_MISC)
+                  if (pf_status.debug >= PF_DEBUG_MISC) {
                       printf("pfsync_input: "
                           "PFSYNC_ACT_UPD_C: "
                           "invalid value\n");
+                  }
                  pfsyncstats.pfsyncs_badval++;
                  continue;
             }

-            bcopy(up->id, &id_key.id, sizeof(id_key.id));
+            bcopy(&up->id, &id_key.id, sizeof(id_key.id));
             id_key.creatorid = up->creatorid;

             st = pf_find_state_byid(&id_key);
             if (st == NULL) {
                  /* We don't have this state. Ask for it. */
-                 switch (pfsync_request_update(up, &pkt->src)) {
-                 case 0:
-                      update_requested = 1;
-                      break;
-                 case ENOMEM:
-                      break;
-                 default:
-                      pfsyncstats.pfsyncs_badstate++;
-                      break;
-                 }
+                 pfsync_request_update(id_key.creatorid, id_key.id);
                  continue;
             }

+            if (ISSET(st->state_flags, PFSTATE_ACK))
+                 pfsync_deferred(st, 1);
+
             sk = st->key[PF_SK_WIRE]; /* XXX right one? */
             sfail = 0;
```

```
-         if (sk->proto == IPPROTO_TCP) {
-               /*
-                * The state should never go backwards except
-                * for syn-proxy states.  Neither should the
-                * sequence window slide backwards.
-                */
-               if (st->src.state > up->src.state &&
-                   (st->src.state < PF_TCPS_PROXY_SRC ||
-                   up->src.state >= PF_TCPS_PROXY_SRC))
-                    sfail = 1;
-               else if (st->dst.state > up->dst.state)
-                    sfail = 2;
-               else if (SEQ_GT(st->src.seqlo, ntohl(up->src.seqlo)))
-                    sfail = 3;
-               else if (st->dst.state >= TCPS_SYN_SENT &&
-                   SEQ_GT(st->dst.seqlo, ntohl(up->dst.seqlo)))
-                    sfail = 4;
-         } else {
+         if (sk->proto == IPPROTO_TCP)
+               sfail = pfsync_upd_tcp(st, &up->src, &up->dst);
+         else {
                /*
-                * Non-TCP protocol state machine always go
-                * forwards
+                * Non-TCP protocol state machine always go forwards
                 */
                if (st->src.state > up->src.state)
                     sfail = 5;
                else if (st->dst.state > up->dst.state)
                     sfail = 6;
          }
-         if (sfail) {
-               if (pf_status.debug >= PF_DEBUG_MISC)
+
+         if (sfail > 0) {
+               if (pf_status.debug >= PF_DEBUG_MISC) {
                     printf("pfsync: ignoring stale update "
                         "(%d) id: %016llx "
                         "creatorid: %08x\n", sfail,
                         betoh64(st->id),
                         ntohl(st->creatorid));
+               }
                pfsyncstats.pfsyncs_stale++;

-               /* we have a better state, send it out */
-               if ((!stale || update_requested) &&
-                   sc->sc_mbuf != NULL) {
-                    pfsync_sendout(sc);
-                    update_requested = 0;
-               }
-               stale++;
-               if (!st->sync_flags)
-                    pfsync_pack_state(PFSYNC_ACT_UPD, st,
-                        PFSYNC_FLAG_STALE);
+               pfsync_update_state(st);
+               schednetisr(NETISR_PFSYNC);
```

```
                continue;
+           } else if (sfail == 0) {
+               pfsync_alloc_scrub_memory(&up->dst, &st->dst);
+               pf_state_peer_ntoh(&up->src, &st->src);
+               pf_state_peer_ntoh(&up->dst, &st->dst);
            }
-           pfsync_alloc_scrub_memory(&up->dst, &st->dst);
-           pf_state_peer_ntoh(&up->src, &st->src);
-           pf_state_peer_ntoh(&up->dst, &st->dst);
            st->expire = ntohl(up->expire) + time_second;
            st->timeout = up->timeout;
+           st->pfsync_time = time_second;
        }
-       if ((update_requested || stale) && sc->sc_mbuf)
-           pfsync_sendout(sc);
        splx(s);

        return (len);
 }

+int pfsync_req_del;
+
 int
-pfsync4_in_del_c(struct pfsync_pkt *pkt, struct mbuf *m, int offset,
int count)
+pfsync_in_ureq(struct pfsync_pkt *pkt, struct mbuf *m, int offset, int
count)
 {
-       struct pfsync_state_del *dp, *dpa;
+       struct pfsync_upd_req *ur, *ura;
+       struct mbuf *mp;
+       int len = count * sizeof(*ur);
+       int i, offp;
+
        struct pf_state_cmp id_key;
        struct pf_state *st;
-       int s;
-
-       struct mbuf *mp;
-       int len = count * sizeof(*dp);
-       int offp;
-       int i;

        mp = m_pulldown(m, offset, len, &offp);
        if (mp == NULL) {
            pfsyncstats.pfsyncs_badlen++;
            return (-1);
        }
-       dpa = (struct pfsync_state_del *)(mp->m_data + offp);
+       ura = (struct pfsync_upd_req *)(mp->m_data + offp);

-       s = splsoftnet();
        for (i = 0; i < count; i++) {
-           dp = &dpa[i];
+           ur = &ura[i];
```

```
-          bcopy(dp->id, &id_key.id, sizeof(id_key.id));
-          id_key.creatorid = dp->creatorid;
+          bcopy(&ur->id, &id_key.id, sizeof(id_key.id));
+          id_key.creatorid = ur->creatorid;

-          st = pf_find_state_byid(&id_key);
-          if (st == NULL) {
-               pfsyncstats.pfsyncs_badstate++;
-               continue;
+          if (id_key.id == 0 && id_key.creatorid == 0)
+               pfsync_bulk_start();
+          else {
+               st = pf_find_state_byid(&id_key);
+               if (st == NULL) {
+                    pfsyncstats.pfsyncs_badstate++;
+                    continue;
+               }
+
+               if (st->timeout == PFTM_UNLINKED)
+                    pfsync_req_del++;
+
+               pfsync_update_state_req(st);
           }
-          st->sync_flags |= PFSTATE_FROMSYNC;
-          pf_unlink_state(st);
      }
-     splx(s);

      return (len);
 }

 int
-pfsync4_in_error(struct pfsync_pkt *pkt, struct mbuf *m, int offset,
int count)
+pfsync_in_del(struct pfsync_pkt *pkt, struct mbuf *m, int offset, int
count)
 {
-     m_freem(m);
-     return (-1);
+     struct mbuf *mp;
+     struct pfsync_state *sa, *sp;
+     struct pf_state_cmp id_key;
+     struct pf_state *st;
+     int len = count * sizeof(*sp);
+     int offp, i;
+     int s;
+
+     mp = m_pulldown(m, offset, len, &offp);
+     if (mp == NULL) {
+          pfsyncstats.pfsyncs_badlen++;
+          return (-1);
+     }
+     sa = (struct pfsync_state *)(mp->m_data + offp);
+
+     s = splsoftnet();
+     for (i = 0; i < count; i++) {
```

```
+           bcopy(sp->id, &id_key.id, sizeof(id_key.id));
+           id_key.creatorid = sp->creatorid;
+
+           st = pf_find_state_byid(&id_key);
+           if (st == NULL) {
+                pfsyncstats.pfsyncs_badstate++;
+                continue;
+           }
+           SET(st->state_flags, PFSTATE_NOSYNC);
+           pf_unlink_state(st);
+      }
+      splx(s);
+
+      return (len);
 }

 int
-pfsync4_in_ureq(struct pfsync_pkt *pkt, struct mbuf *m, int offset, int
count)
+pfsync_in_del_c(struct pfsync_pkt *pkt, struct mbuf *m, int offset, int
count)
 {
-      struct pfsync_softc *sc = pfsyncif;
-      struct pfsync_state_upd_req *rup, *rupa;
+      struct mbuf *mp;
+      struct pfsync_del_c *sa, *sp;
       struct pf_state_cmp id_key;
       struct pf_state *st;
+      int len = count * sizeof(*sp);
+      int offp, i;
       int s;

-      struct mbuf *mp;
-      int len = count * sizeof(*rup);
-      int offp;
-      int i;
-
       mp = m_pulldown(m, offset, len, &offp);
       if (mp == NULL) {
            pfsyncstats.pfsyncs_badlen++;
            return (-1);
       }
-      rupa = (struct pfsync_state_upd_req *)(mp->m_data + offp);
+      sa = (struct pfsync_del_c *)(mp->m_data + offp);

       s = splsoftnet();
-      if (sc->sc_mbuf != NULL)
-           pfsync_sendout(sc);
-
       for (i = 0; i < count; i++) {
-               rup = &rupa[i];
+           sp = &sa[i];

-           bcopy(rup->id, &id_key.id, sizeof(id_key.id));
-           id_key.creatorid = rup->creatorid;
+           bcopy(&sp->id, &id_key.id, sizeof(id_key.id));
+           id_key.creatorid = sp->creatorid;

-           if (id_key.id == 0 && id_key.creatorid == 0) {
-                sc->sc_ureq_received = time_uptime;
-                if (sc->sc_bulk_send_next == NULL)
-                     sc->sc_bulk_send_next =
-                         TAILQ_FIRST(&state_list);
-                sc->sc_bulk_terminator = sc->sc_bulk_send_next;
-                if (pf_status.debug >= PF_DEBUG_MISC)
-                     printf("pfsync: received "
-                         "bulk update request\n");
-                pfsync_send_bus(sc, PFSYNC_BUS_START);
-                timeout_add_sec(&sc->sc_bulk_tmo, 1);
-           } else {
-                st = pf_find_state_byid(&id_key);
-                if (st == NULL) {
-                     pfsyncstats.pfsyncs_badstate++;
-                     continue;
-                }
-                if (!st->sync_flags)
-                     pfsync_pack_state(PFSYNC_ACT_UPD, st, 0);
+           st = pf_find_state_byid(&id_key);
+           if (st == NULL) {
+                pfsyncstats.pfsyncs_badstate++;
+                continue;
            }
-      }

-      if (sc->sc_mbuf != NULL)
-           pfsync_sendout(sc);
+           SET(st->state_flags, PFSTATE_NOSYNC);
+           pf_unlink_state(st);
+      }
       splx(s);

       return (len);
 }

 int
-pfsync4_in_bus(struct pfsync_pkt *pkt, struct mbuf *m, int offset, int
count)
+pfsync_in_bus(struct pfsync_pkt *pkt, struct mbuf *m, int offset, int
count)
 {
       struct pfsync_softc *sc = pfsyncif;
-      struct pfsync_state_bus *bus;
-
+      struct pfsync_bus *bus;
       struct mbuf *mp;
+      int len = count * sizeof(*bus);
       int offp;

       /* If we're not waiting for a bulk update, who cares. */
       if (sc->sc_ureq_sent == 0)
-           return (sizeof(*bus));
+           return (len);
```

```
-	mp = m_pulldown(m, offset, sizeof(*bus), &offp);
+	mp = m_pulldown(m, offset, len, &offp);
 	if (mp == NULL) {
 		pfsyncstats.pfsyncs_badlen++;
 		return (-1);
 	}
-	bus = (struct pfsync_state_bus *)(mp->m_data + offp);
+	bus = (struct pfsync_bus *)(mp->m_data + offp);

 	switch (bus->status) {
 	case PFSYNC_BUS_START:
-		timeout_add(&sc->sc_bulkfail_tmo,
+		timeout_add_sec(&sc->sc_bulkfail_tmo, 5); /* XXX magic */
+#if XXX
 		    pf_pool_limits[PF_LIMIT_STATES].limit /
 		    (PFSYNC_BULKPACKETS * sc->sc_maxcount));
+#endif
 		if (pf_status.debug >= PF_DEBUG_MISC)
-			printf("pfsync: received bulk "
-			    "update start\n");
+			printf("pfsync: received bulk update start\n");
 		break;
+
 	case PFSYNC_BUS_END:
 		if (time_uptime - ntohl(bus->endtime) >=
 		    sc->sc_ureq_sent) {
@@ -1092,33 +1251,31 @@ pfsync4_in_bus(struct pfsync_pkt *pkt, s
 		break;
 	}

-	return (sizeof(*bus));
+	return (len);
 }

 int
-pfsync4_in_tdb_upd(struct pfsync_pkt *pkt, struct mbuf *m, int offset,
-    int count)
+pfsync_in_tdb(struct pfsync_pkt *pkt, struct mbuf *m, int offset, int
count)
 {
-	struct pfsync_tdb *pt;
-	int len = count * sizeof(*pt);
-
-#ifdef IPSEC
-	int s;
+	int len = count * sizeof(struct pfsync_tdb);

+#if 0 && defined(IPSEC)
+	struct pfsync_tdb *tp;
 	struct mbuf *mp;
 	int offp;
 	int i;
+	int s;

 	mp = m_pulldown(m, offset, len, &offp);
	if (mp == NULL) {
		pfsyncstats.pfsyncs_badlen++;
		return (-1);
	}
-	pt = (struct pfsync_tdb *)(mp->m_data + offp);
+	tp = (struct pfsync_tdb *)(mp->m_data + offp);

	s = splsoftnet();
	for (i = 0; i < count; i++)
-		pfsync_update_net_tdb(&pt[i]);
+		pfsync_update_net_tdb(&tp[i]); /* XXX */
	splx(s);
 #endif

@@ -1126,6 +1283,27 @@ pfsync4_in_tdb_upd(struct pfsync_pkt *pk
 }

 int
+pfsync_in_eof(struct pfsync_pkt *pkt, struct mbuf *m, int offset, int
count)
+{
+	/* check if we are at the right place in the packet */
+	if (offset != m->m_pkthdr.len - sizeof(struct pfsync_eof))
+		pfsyncstats.pfsyncs_badact++;
+
+	/* we're done. free and let the caller return */
+	m_freem(m);
+	return (-1);
+}
+
+int
+pfsync_in_error(struct pfsync_pkt *pkt, struct mbuf *m, int offset, int
count)
+{
+	pfsyncstats.pfsyncs_badact++;
+
+	m_freem(m);
+	return (-1);
+}
+
+int
 pfsyncoutput(struct ifnet *ifp, struct mbuf *m, struct sockaddr *dst,
	struct rtentry *rt)
 {
@@ -1143,12 +1321,15 @@ pfsyncioctl(struct ifnet *ifp, u_long cm
	struct ip_moptions *imo = &sc->sc_imo;
	struct pfsyncreq pfsyncr;
	struct ifnet    *sifp;
+	struct ip *ip;
	int s, error;

	switch (cmd) {
+#if 0
	case SIOCSIFADDR:
	case SIOCAIFADDR:
	case SIOCSIFDSTADDR:
```

```
+#endif
 	case SIOCSIFFLAGS:
 		if (ifp->if_flags & IFF_UP)
 			ifp->if_flags |= IFF_RUNNING;
@@ -1156,26 +1337,27 @@ pfsyncioctl(struct ifnet *ifp, u_long cm
 			ifp->if_flags &= ~IFF_RUNNING;
 		break;
 	case SIOCSIFMTU:
-		if (ifr->ifr_mtu < PFSYNC_MINMTU)
+		if (ifr->ifr_mtu <= PFSYNC_MINPKT)
 			return (EINVAL);
-		if (ifr->ifr_mtu > MCLBYTES)
+		if (ifr->ifr_mtu > MCLBYTES) /* XXX could be bigger */
 			ifr->ifr_mtu = MCLBYTES;
-		s = splnet();
-		if (ifr->ifr_mtu < ifp->if_mtu)
-			pfsync_sendout(sc);
-		pfsync_setmtu(sc, ifr->ifr_mtu);
-		splx(s);
+		if (ifr->ifr_mtu < ifp->if_mtu) {
+			s = splnet();
+			pfsync_sendout();
+			splx(s);
+		}
+		ifp->if_mtu = ifr->ifr_mtu;
 		break;
 	case SIOCGETPFSYNC:
 		bzero(&pfsyncr, sizeof(pfsyncr));
-		if (sc->sc_sync_ifp)
+		if (sc->sc_sync_if) {
 			strlcpy(pfsyncr.pfsyncr_syncdev,
-			    sc->sc_sync_ifp->if_xname, IFNAMSIZ);
+			    sc->sc_sync_if->if_xname, IFNAMSIZ);
+		}
 		pfsyncr.pfsyncr_syncpeer = sc->sc_sync_peer;
 		pfsyncr.pfsyncr_maxupdates = sc->sc_maxupdates;
-		if ((error = copyout(&pfsyncr, ifr->ifr_data,
sizeof(pfsyncr))))
-			return (error);
-		break;
+		return (copyout(&pfsyncr, ifr->ifr_data, sizeof(pfsyncr)));
+
 	case SIOCSETPFSYNC:
 		if ((error = suser(p, p->p_acflag)) != 0)
 			return (error);
@@ -1193,17 +1375,10 @@ pfsyncioctl(struct ifnet *ifp, u_long cm
 		sc->sc_maxupdates = pfsyncr.pfsyncr_maxupdates;
 
 		if (pfsyncr.pfsyncr_syncdev[0] == 0) {
-			sc->sc_sync_ifp = NULL;
-			if (sc->sc_mbuf_net != NULL) {
-				/* Don't keep stale pfsync packets around. */
-				s = splnet();
-				m_freem(sc->sc_mbuf_net);
-				sc->sc_mbuf_net = NULL;
-				sc->sc_statep_net.s = NULL;
```

```
-                       splx(s);
-               }
+               sc->sc_sync_if = NULL;
                if (imo->imo_num_memberships > 0) {
-                       in_delmulti(imo->imo_membership[--imo-
>imo_num_memberships]);
+                       in_delmulti(imo->imo_membership[
+                           --imo->imo_num_memberships]);
                        imo->imo_multicast_ifp = NULL;
                }
                break;
@@ -1214,25 +1389,23 @@ pfsyncioctl(struct ifnet *ifp, u_long cm

        s = splnet();
        if (sifp->if_mtu < sc->sc_if.if_mtu ||
-           (sc->sc_sync_ifp != NULL &&
-           sifp->if_mtu < sc->sc_sync_ifp->if_mtu) ||
+           (sc->sc_sync_if != NULL &&
+           sifp->if_mtu < sc->sc_sync_if->if_mtu) ||
            sifp->if_mtu < MCLBYTES - sizeof(struct ip))
-               pfsync_sendout(sc);
-       sc->sc_sync_ifp = sifp;
-
-       pfsync_setmtu(sc, sc->sc_if.if_mtu);
+               pfsync_sendout();
+       sc->sc_sync_if = sifp;

        if (imo->imo_num_memberships > 0) {
                in_delmulti(imo->imo_membership[--imo-
>imo_num_memberships]);
                imo->imo_multicast_ifp = NULL;
        }

-       if (sc->sc_sync_ifp &&
+       if (sc->sc_sync_if &&
            sc->sc_sync_peer.s_addr == INADDR_PFSYNC_GROUP) {
                struct in_addr addr;

-               if (!(sc->sc_sync_ifp->if_flags & IFF_MULTICAST)) {
-                       sc->sc_sync_ifp = NULL;
+               if (!(sc->sc_sync_if->if_flags & IFF_MULTICAST)) {
+                       sc->sc_sync_if = NULL;
                        splx(s);
                        return (EADDRNOTAVAIL);
                }
@@ -1240,19 +1413,31 @@ pfsyncioctl(struct ifnet *ifp, u_long cm
                addr.s_addr = INADDR_PFSYNC_GROUP;

                if ((imo->imo_membership[0] =
-                   in_addmulti(&addr, sc->sc_sync_ifp)) == NULL) {
-                       sc->sc_sync_ifp = NULL;
+                   in_addmulti(&addr, sc->sc_sync_if)) == NULL) {
+                       sc->sc_sync_if = NULL;
                        splx(s);
                        return (ENOBUFS);
                }
                imo->imo_num_memberships++;
-               imo->imo_multicast_ifp = sc->sc_sync_ifp;
+               imo->imo_multicast_ifp = sc->sc_sync_if;
                imo->imo_multicast_ttl = PFSYNC_DFLTTL;
                imo->imo_multicast_loop = 0;
        }

-       if (sc->sc_sync_ifp ||
-           sc->sc_sendaddr.s_addr != INADDR_PFSYNC_GROUP) {
+       ip = &sc->sc_template;
+       bzero(ip, sizeof(*ip));
+       ip->ip_v = IPVERSION;
+       ip->ip_hl = sizeof(sc->sc_template) >> 2;
+       ip->ip_tos = IPTOS_LOWDELAY;
+       /* len and id are set later */
+       ip->ip_off = htons(IP_DF);
+       ip->ip_ttl = PFSYNC_DFLTTL;
+       ip->ip_p = IPPROTO_PFSYNC;
+       ip->ip_src.s_addr = INADDR_ANY;
+       ip->ip_dst.s_addr = sc->sc_sync_peer.s_addr;
+
+       if (sc->sc_sync_if) {
+#if 0
                /* Request a full state table update. */
                sc->sc_ureq_sent = time_uptime;
 #if NCARP > 0
@@ -1263,12 +1448,9 @@ pfsyncioctl(struct ifnet *ifp, u_long cm
                if (pf_status.debug >= PF_DEBUG_MISC)
                    printf("pfsync: requesting bulk update\n");
                timeout_add_sec(&sc->sc_bulkfail_tmo, 5);
-               error = pfsync_request_update(NULL, NULL);
-               if (error == ENOMEM) {
-                   splx(s);
-                   return (ENOMEM);
-               }
-               pfsync_sendout(sc);
+               /* XXX bulks done this way? */
+               pfsync_request_update(0, 0);
+#endif
        }
        splx(s);

@@ -1281,691 +1463,683 @@ pfsyncioctl(struct ifnet *ifp, u_long cm
    return (0);
 }

-void
-pfsync_setmtu(struct pfsync_softc *sc, int mtu_req)
+int
+pfsync_out_state(struct pf_state *st, struct mbuf *m, int offset)
 {
-    int mtu;
+    struct pfsync_state *sp = (struct pfsync_state *)(m->m_data +
offset);

-    if (sc->sc_sync_ifp && sc->sc_sync_ifp->if_mtu < mtu_req)
-           mtu = sc->sc_sync_ifp->if_mtu;
-       else
-           mtu = mtu_req;
+       pfsync_state_export(sp, st);

-       sc->sc_maxcount = (mtu - sizeof(struct pfsync_header)) /
-           sizeof(struct pfsync_state);
-       if (sc->sc_maxcount > 254)
-           sc->sc_maxcount = 254;
-       sc->sc_if.if_mtu = sizeof(struct pfsync_header) +
-           sc->sc_maxcount * sizeof(struct pfsync_state);
+       return (sizeof(*sp));
 }

-struct mbuf *
-pfsync_get_mbuf(struct pfsync_softc *sc, u_int8_t action, void **sp)
+int
+pfsync_out_iack(struct pf_state *st, struct mbuf *m, int offset)
 {
-       struct pfsync_header *h;
-       struct mbuf *m;
-       int len;
+       struct pfsync_ins_ack *iack =
+           (struct pfsync_ins_ack *)(m->m_data + offset);

-       MGETHDR(m, M_DONTWAIT, MT_DATA);
-       if (m == NULL) {
-           sc->sc_if.if_oerrors++;
-           return (NULL);
-       }
+       iack->id = st->id;
+       iack->creatorid = st->creatorid;

-       switch (action) {
-       case PFSYNC_ACT_CLR:
-           len = sizeof(struct pfsync_header) +
-               sizeof(struct pfsync_state_clr);
-           break;
-       case PFSYNC_ACT_UPD_C:
-           len = (sc->sc_maxcount * sizeof(struct pfsync_state_upd)) +
-               sizeof(struct pfsync_header);
-           break;
-       case PFSYNC_ACT_DEL_C:
-           len = (sc->sc_maxcount * sizeof(struct pfsync_state_del)) +
-               sizeof(struct pfsync_header);
-           break;
-       case PFSYNC_ACT_UREQ:
-           len = (sc->sc_maxcount * sizeof(struct pfsync_state_upd_req))
+
-               sizeof(struct pfsync_header);
-           break;
-       case PFSYNC_ACT_BUS:
-           len = sizeof(struct pfsync_header) +
-               sizeof(struct pfsync_state_bus);
-           break;
-       case PFSYNC_ACT_TDB_UPD:
```

```
-           len = (sc->sc_maxcount * sizeof(struct pfsync_tdb)) +
-               sizeof(struct pfsync_header);
-           break;
-       default:
-           len = (sc->sc_maxcount * sizeof(struct pfsync_state)) +
-               sizeof(struct pfsync_header);
-           break;
-       }
+       return (sizeof(*iack));
+}

-       if (len > MHLEN) {
-           MCLGET(m, M_DONTWAIT);
-           if ((m->m_flags & M_EXT) == 0) {
-               m_free(m);
-               sc->sc_if.if_oerrors++;
-               return (NULL);
-           }
-           m->m_data += (MCLBYTES - len) &~ (sizeof(long) - 1);
-       } else
-           MH_ALIGN(m, len);
+int
+pfsync_out_upd_c(struct pf_state *st, struct mbuf *m, int offset)
+{
+       struct pfsync_upd_c *up = (struct pfsync_upd_c *)(m->m_data +
offset);
+
+       up->id = st->id;
+       pf_state_peer_hton(&st->src, &up->src);
+       pf_state_peer_hton(&st->dst, &up->dst);
+       up->creatorid = st->creatorid;

-       m->m_pkthdr.rcvif = NULL;
-       m->m_pkthdr.len = m->m_len = sizeof(struct pfsync_header);
-       h = mtod(m, struct pfsync_header *);
-       h->version = PFSYNC_VERSION;
-       h->af = 0;
-       h->count = 0;
-       h->action = action;
-       if (action != PFSYNC_ACT_TDB_UPD)
-           bcopy(&pf_status.pf_chksum, &h->pf_chksum,
-               PF_MD5_DIGEST_LENGTH);
-
-       *sp = (void *)((char *)h + PFSYNC_HDRLEN);
-       if (action == PFSYNC_ACT_TDB_UPD)
-           timeout_add_sec(&sc->sc_tdb_tmo, 1);
+       up->expire = pf_state_expires(st);
+       if (up->expire <= time_second)
+           up->expire = htonl(0);
        else
-           timeout_add_sec(&sc->sc_tmo, 1);
-       return (m);
+           up->expire = htonl(up->expire - time_second);
+       up->timeout = st->timeout;
+
+       bzero(up->_pad, sizeof(up->_pad)); /* XXX */
```

```
+
+	return (sizeof(*up));
 }

 int
-pfsync_pack_state(u_int8_t action, struct pf_state *st, int flags)
+pfsync_out_del(struct pf_state *st, struct mbuf *m, int offset)
 {
-	struct ifnet *ifp = NULL;
-	struct pfsync_softc *sc = pfsyncif;
-	struct pfsync_header *h, *h_net;
-	struct pfsync_state *sp = NULL;
-	struct pfsync_state_upd *up = NULL;
-	struct pfsync_state_del *dp = NULL;
-	int s, ret = 0;
-	u_int8_t i = 255, newaction = 0;
+	struct pfsync_del_c *dp = (struct pfsync_del_c *)(m->m_data +
offset);

-	if (sc == NULL)
-		return (0);
-	ifp = &sc->sc_if;
+	dp->id = st->id;
+	dp->creatorid = st->creatorid;

-	/*
-	 * If a packet falls in the forest and there's nobody around to
-	 * hear, does it make a sound?
-	 */
-	if (ifp->if_bpf == NULL && sc->sc_sync_ifp == NULL &&
-	    sc->sc_sync_peer.s_addr == INADDR_PFSYNC_GROUP) {
-		/* Don't leave any stale pfsync packets hanging around. */
-		if (sc->sc_mbuf != NULL) {
-			m_freem(sc->sc_mbuf);
-			sc->sc_mbuf = NULL;
-			sc->sc_statep.s = NULL;
+	SET(st->state_flags, PFSTATE_NOSYNC);
+
+	return (sizeof(*dp));
+}
+
+void
+pfsync_drop(struct pfsync_softc *sc)
+{
+	struct pf_state *st;
+	struct pfsync_upd_req_item *ur;
+	struct tdb *t;
+	int q;
+
+	for (q = 0; q < PFSYNC_S_COUNT; q++) {
+		if (TAILQ_EMPTY(&sc->sc_qs[q]))
+			continue;
+
+		TAILQ_FOREACH(st, &sc->sc_qs[q], sync_list) {
+#ifdef PFSYNC_DEBUG
+			KASSERT(st->sync_state == q);
```

```
+#endif
+           st->sync_state = PFSYNC_S_NONE;
        }
-       return (0);
+       TAILQ_INIT(&sc->sc_qs[q]);
    }

-   if (action >= PFSYNC_ACT_MAX)
-       return (EINVAL);
+   while ((ur = TAILQ_FIRST(&sc->sc_upd_req_list)) != NULL) {
+       TAILQ_REMOVE(&sc->sc_upd_req_list, ur, ur_entry);
+       pool_put(&sc->sc_pool, ur);
+   }

-   s = splnet();
-   if (sc->sc_mbuf == NULL) {
-       if ((sc->sc_mbuf = pfsync_get_mbuf(sc, action,
-          (void *)&sc->sc_statep.s)) == NULL) {
-           splx(s);
-           return (ENOMEM);
-       }
-       h = mtod(sc->sc_mbuf, struct pfsync_header *);
-   } else {
-       h = mtod(sc->sc_mbuf, struct pfsync_header *);
-       if (h->action != action) {
-           pfsync_sendout(sc);
-           if ((sc->sc_mbuf = pfsync_get_mbuf(sc, action,
-              (void *)&sc->sc_statep.s)) == NULL) {
-               splx(s);
-               return (ENOMEM);
-           }
-           h = mtod(sc->sc_mbuf, struct pfsync_header *);
-       } else {
-           /*
-            * If it's an update, look in the packet to see if
-            * we already have an update for the state.
-            */
-           if (action == PFSYNC_ACT_UPD && sc->sc_maxupdates) {
-               struct pfsync_state *usp =
-                   (void *)((char *)h + PFSYNC_HDRLEN);
-
-               for (i = 0; i < h->count; i++) {
-                   if (!memcmp(usp->id, &st->id,
-                      PFSYNC_ID_LEN) &&
-                      usp->creatorid == st->creatorid) {
-                       sp = usp;
-                       sp->updates++;
-                       break;
-                   }
-                   usp++;
-               }
-           }
-       }
+   sc->sc_plus = NULL;
+
+   if (!TAILQ_EMPTY(&sc->sc_tdb_q)) {
```

```
+		TAILQ_FOREACH(t, &sc->sc_tdb_q, tdb_sync_entry)
+			CLR(t->tdb_flags, TDBF_PFSYNC);
+
+		TAILQ_INIT(&sc->sc_tdb_q);
	}

-	st->pfsync_time = time_uptime;
+	sc->sc_len = PFSYNC_MINPKT;
+}
+
+void
+pfsync_sendout(void)
+{
+	struct pfsync_softc *sc = pfsyncif;
+#if NBPFILTER > 0
+	struct ifnet *ifp = &sc->sc_if;
+#endif
+	struct mbuf *m;
+	struct ip *ip;
+	struct pfsync_header *ph;
+	struct pfsync_subheader *subh;
+	struct pf_state *st;
+	struct pfsync_upd_req_item *ur;
+	struct tdb *t;

-	if (sp == NULL) {
-		/* not a "duplicate" update */
-		i = 255;
-		sp = sc->sc_statep.s++;
-		sc->sc_mbuf->m_pkthdr.len =
-		    sc->sc_mbuf->m_len += sizeof(struct pfsync_state);
-		h->count++;
-		bzero(sp, sizeof(*sp));
+	int offset;
+	int q, count = 0;

-		pfsync_state_export(sp, st);
+	splassert(IPL_NET);

-		if (flags & PFSYNC_FLAG_STALE)
-			sp->sync_flags |= PFSTATE_STALE;
-	} else {
-		pf_state_peer_hton(&st->src, &sp->src);
-		pf_state_peer_hton(&st->dst, &sp->dst);
+	if (sc == NULL || sc->sc_len == PFSYNC_MINPKT)
+		return;

-		if (st->expire <= time_second)
-			sp->expire = htonl(0);
-		else
-			sp->expire = htonl(st->expire - time_second);
+	MGETHDR(m, M_DONTWAIT, MT_DATA);
+	if (m == NULL) {
+		sc->sc_if.if_oerrors++;
+		pfsyncstats.pfsyncs_onomem++;
+		pfsync_drop(sc);
```

```
+           return;
      }

-     /* do we need to build "compressed" actions for network transfer?
*/
-     if (sc->sc_sync_ifp && flags & PFSYNC_FLAG_COMPRESS) {
-           switch (action) {
-           case PFSYNC_ACT_UPD:
-                 newaction = PFSYNC_ACT_UPD_C;
-                 break;
-           case PFSYNC_ACT_DEL:
-                 newaction = PFSYNC_ACT_DEL_C;
-                 break;
-           default:
-                 /* by default we just send the uncompressed states */
-                 break;
+     if (max_linkhdr + sc->sc_len > MHLEN) {
+           MCLGETI(m, M_DONTWAIT, NULL, max_linkhdr + sc->sc_len);
+           if (!ISSET(m->m_flags, M_EXT)) {
+                 m_free(m);
+                 sc->sc_if.if_oerrors++;
+                 pfsyncstats.pfsyncs_onomem++;
+                 pfsync_drop(sc);
+                 return;
            }
      }
+     m->m_data += max_linkhdr;
+     m->m_len = m->m_pkthdr.len = sc->sc_len;

-     if (newaction) {
-           if (sc->sc_mbuf_net == NULL) {
-                 if ((sc->sc_mbuf_net = pfsync_get_mbuf(sc, newaction,
-                     (void *)&sc->sc_statep_net.s)) == NULL) {
-                       splx(s);
-                       return (ENOMEM);
-                 }
-           }
-           h_net = mtod(sc->sc_mbuf_net, struct pfsync_header *);
+     /* build the ip header */
+     ip = (struct ip *)m->m_data;
+     bcopy(&sc->sc_template, ip, sizeof(*ip));
+     offset = sizeof(*ip);
+
+     ip->ip_len = htons(m->m_pkthdr.len);
+     ip->ip_id = htons(ip_randomid());
+
+     /* build the pfsync header */
+     ph = (struct pfsync_header *)(m->m_data + offset);
+     bzero(ph, sizeof(*ph));
+     offset += sizeof(*ph);

-           switch (newaction) {
-           case PFSYNC_ACT_UPD_C:
-                 if (i != 255) {
-                       up = (void *)((char *)h_net +
-                           PFSYNC_HDRLEN + (i * sizeof(*up)));
-                       up->updates++;
-               } else {
-                       h_net->count++;
-                       sc->sc_mbuf_net->m_pkthdr.len =
-                           sc->sc_mbuf_net->m_len += sizeof(*up);
-                       up = sc->sc_statep_net.u++;
-
-                       bzero(up, sizeof(*up));
-                       bcopy(&st->id, up->id, sizeof(up->id));
-                       up->creatorid = st->creatorid;
-               }
-               up->timeout = st->timeout;
-               up->expire = sp->expire;
-               up->src = sp->src;
-               up->dst = sp->dst;
-               break;
-       case PFSYNC_ACT_DEL_C:
-               sc->sc_mbuf_net->m_pkthdr.len =
-                   sc->sc_mbuf_net->m_len += sizeof(*dp);
-               dp = sc->sc_statep_net.d++;
-               h_net->count++;
-
-               bzero(dp, sizeof(*dp));
-               bcopy(&st->id, dp->id, sizeof(dp->id));
-               dp->creatorid = st->creatorid;
-               break;
+       ph->version = PFSYNC_VERSION;
+       ph->len = htons(sc->sc_len - sizeof(*ip));
+       bcopy(pf_status.pf_chksum, ph->pfcksum, PF_MD5_DIGEST_LENGTH);
+
+       /* walk the queues */
+       for (q = 0; q < PFSYNC_S_COUNT; q++) {
+               if (TAILQ_EMPTY(&sc->sc_qs[q]))
+                       continue;
+
+               subh = (struct pfsync_subheader *)(m->m_data + offset);
+               offset += sizeof(*subh);
+
+               count = 0;
+               TAILQ_FOREACH(st, &sc->sc_qs[q], sync_list) {
+#ifdef PFSYNC_DEBUG
+                       KASSERT(st->sync_state == q);
+#endif
+
+                       offset += pfsync_qs[q].write(st, m, offset);
+                       st->sync_state = PFSYNC_S_NONE;
+                       count++;
                }
+               TAILQ_INIT(&sc->sc_qs[q]);
+
+               bzero(subh, sizeof(*subh));
+               subh->action = pfsync_qs[q].action;
+               subh->count = htons(count);
        }

-       if (h->count == sc->sc_maxcount ||
-        (sc->sc_maxupdates && (sp->updates >= sc->sc_maxupdates)))
-         ret = pfsync_sendout(sc);
+    if (!TAILQ_EMPTY(&sc->sc_upd_req_list)) {
+         subh = (struct pfsync_subheader *)(m->m_data + offset);
+         offset += sizeof(*subh);

-    splx(s);
-    return (ret);
-}
+         count = 0;
+         while ((ur = TAILQ_FIRST(&sc->sc_upd_req_list)) != NULL) {
+              TAILQ_REMOVE(&sc->sc_upd_req_list, ur, ur_entry);

-/* This must be called in splnet() */
-int
-pfsync_request_update(struct pfsync_state_upd *up, struct in_addr *src)
-{
-    struct pfsync_header *h;
-    struct pfsync_softc *sc = pfsyncif;
-    struct pfsync_state_upd_req *rup;
-    int ret = 0;
+              bcopy(&ur->ur_msg, m->m_data + offset,
+                  sizeof(ur->ur_msg));
+              offset += sizeof(ur->ur_msg);

-    if (sc == NULL)
-         return (0);
+              pool_put(&sc->sc_pool, ur);

-    if (sc->sc_mbuf == NULL) {
-         if ((sc->sc_mbuf = pfsync_get_mbuf(sc, PFSYNC_ACT_UREQ,
-             (void *)&sc->sc_statep.s)) == NULL)
-              return (ENOMEM);
-         h = mtod(sc->sc_mbuf, struct pfsync_header *);
-    } else {
-         h = mtod(sc->sc_mbuf, struct pfsync_header *);
-         if (h->action != PFSYNC_ACT_UREQ) {
-              pfsync_sendout(sc);
-              if ((sc->sc_mbuf = pfsync_get_mbuf(sc, PFSYNC_ACT_UREQ,
-                  (void *)&sc->sc_statep.s)) == NULL)
-                   return (ENOMEM);
-              h = mtod(sc->sc_mbuf, struct pfsync_header *);
+              count++;
         }
-    }

-    if (src != NULL)
-         sc->sc_sendaddr = *src;
-    sc->sc_mbuf->m_pkthdr.len = sc->sc_mbuf->m_len += sizeof(*rup);
-    h->count++;
-    rup = sc->sc_statep.r++;
-    bzero(rup, sizeof(*rup));
-    if (up != NULL) {
-         bcopy(up->id, rup->id, sizeof(rup->id));
-         rup->creatorid = up->creatorid;
+         bzero(subh, sizeof(*subh));
```

```
+		subh->action = PFSYNC_ACT_UPD_REQ;
+		subh->count = htons(count);
	}

-	if (h->count == sc->sc_maxcount)
-		ret = pfsync_sendout(sc);
+	/* has someone built a custom region for us to add? */
+	if (sc->sc_plus != NULL) {
+		bcopy(sc->sc_plus, m->m_data + offset, sc->sc_pluslen);
+		offset += sc->sc_pluslen;

-	return (ret);
-}
+		sc->sc_plus = NULL;
+	}

-int
-pfsync_clear_states(u_int32_t creatorid, char *ifname)
-{
-	struct pfsync_softc *sc = pfsyncif;
-	struct pfsync_state_clr *cp;
-	int s, ret;
+	if (!TAILQ_EMPTY(&sc->sc_tdb_q)) {
+		subh = (struct pfsync_subheader *)(m->m_data + offset);
+		offset += sizeof(*subh);

-	if (sc == NULL)
-		return (0);
+		count = 0;
+		TAILQ_FOREACH(t, &sc->sc_tdb_q, tdb_sync_entry) {
+			offset += pfsync_out_tdb(t, m, offset);
+			CLR(t->tdb_flags, TDBF_PFSYNC);

-	s = splnet();
-	if (sc->sc_mbuf != NULL)
-		pfsync_sendout(sc);
-	if ((sc->sc_mbuf = pfsync_get_mbuf(sc, PFSYNC_ACT_CLR,
-	    (void *)&sc->sc_statep.c)) == NULL) {
-		splx(s);
-		return (ENOMEM);
+			count++;
+		}
+		TAILQ_INIT(&sc->sc_tdb_q);
+
+		bzero(subh, sizeof(*subh));
+		subh->action = PFSYNC_ACT_TDB;
+		subh->count = htons(count);
	}
-	sc->sc_mbuf->m_pkthdr.len = sc->sc_mbuf->m_len += sizeof(*cp);
-	cp = sc->sc_statep.c;
-	cp->creatorid = creatorid;
-	if (ifname != NULL)
-		strlcpy(cp->ifname, ifname, IFNAMSIZ);

-	ret = (pfsync_sendout(sc));
-	splx(s);
```

```
-	return (ret);
-}
+	subh = (struct pfsync_subheader *)(m->m_data + offset);
+	offset += sizeof(*subh);

-void
-pfsync_timeout(void *v)
-{
-	struct pfsync_softc *sc = v;
-	int s;
+	bzero(subh, sizeof(*subh));
+	subh->action = PFSYNC_ACT_EOF;
+	subh->count = htons(1);

-	s = splnet();
-	pfsync_sendout(sc);
-	splx(s);
+	/* XXX write checksum in EOF here */
+
+	/* we're done, let's put it on the wire */
+#if NBPFILTER > 0
+	if (ifp->if_bpf) {
+		m->m_data += sizeof(*ip);
+		m->m_len = m->m_pkthdr.len = sc->sc_len - sizeof(*ip);
+		bpf_mtap(ifp->if_bpf, m, BPF_DIRECTION_OUT);
+		m->m_data -= sizeof(*ip);
+		m->m_len = m->m_pkthdr.len = sc->sc_len;
+	}
+#endif
+	sc->sc_if.if_opackets++;
+	sc->sc_if.if_obytes += m->m_pkthdr.len;
+
+	if (ip_output(m, NULL, NULL, IP_RAWOUTPUT, &sc->sc_imo, NULL) == 0)
+		pfsyncstats.pfsyncs_opackets++;
+	else
+		pfsyncstats.pfsyncs_oerrors++;
+
+	/* start again */
+	sc->sc_len = PFSYNC_MINPKT;
 }

 void
-pfsync_tdb_timeout(void *v)
+pfsync_insert_state(struct pf_state *st)
 {
-	struct pfsync_softc *sc = v;
-	int s;
+	struct pfsync_softc *sc = pfsyncif;

-	s = splnet();
-	pfsync_tdb_sendout(sc);
-	splx(s);
+	splassert(IPL_SOFTNET);
+
+	if (ISSET(st->rule.ptr->rule_flag, PFRULE_NOSYNC) ||
+	    st->key[PF_SK_WIRE]->proto == IPPROTO_PFSYNC) {
```

```
+		SET(st->state_flags, PFSTATE_NOSYNC);
+		return;
+	}
+
+	if (sc == NULL || ISSET(st->state_flags, PFSTATE_NOSYNC))
+		return;
+
+#ifdef PFSYNC_DEBUG
+	KASSERT(st->sync_state == PFSYNC_S_NONE);
+#endif
+
+	if (sc->sc_len == PFSYNC_MINPKT)
+		timeout_add_sec(&sc->sc_tmo, 1);
+
+	pfsync_q_ins(st, PFSYNC_S_INS);
+
+	if (ISSET(st->state_flags, PFSTATE_ACK))
+		schednetisr(NETISR_PFSYNC);
+	else
+		st->sync_updates = 0;
+}
+
+int defer = 10;
+
+int
+pfsync_defer(struct pf_state *st, struct mbuf *m)
+{
+	struct pfsync_softc *sc = pfsyncif;
+	struct pfsync_deferral *pd;
+
+	splassert(IPL_SOFTNET);
+
+	if (sc->sc_deferred >= 128)
+		pfsync_undefer(TAILQ_FIRST(&sc->sc_deferrals), 0);
+
+	pd = pool_get(&sc->sc_pool, M_NOWAIT);
+	if (pd == NULL)
+		return (0);
+	sc->sc_deferred++;
+
+	m->m_pkthdr.pf.flags |= PF_TAG_GENERATED;
+	SET(st->state_flags, PFSTATE_ACK);
+
+	pd->pd_st = st;
+	pd->pd_m = m;
+
+	TAILQ_INSERT_TAIL(&sc->sc_deferrals, pd, pd_entry);
+	timeout_set(&pd->pd_tmo, pfsync_defer_tmo, pd);
+	timeout_add(&pd->pd_tmo, defer);
+
+	return (1);
 }

-/* This must be called in splnet() */
 void
-pfsync_send_bus(struct pfsync_softc *sc, u_int8_t status)
```

```
+pfsync_undefer(struct pfsync_deferral *pd, int drop)
 {
-	struct pfsync_state_bus *bus;
+	struct pfsync_softc *sc = pfsyncif;
+	int s;
+
+	splassert(IPL_SOFTNET);

-	if (sc->sc_mbuf != NULL)
-		pfsync_sendout(sc);
+	TAILQ_REMOVE(&sc->sc_deferrals, pd, pd_entry);
+	sc->sc_deferred--;

-	if (pfsync_sync_ok &&
-	    (sc->sc_mbuf = pfsync_get_mbuf(sc, PFSYNC_ACT_BUS,
-	    (void *)&sc->sc_statep.b)) != NULL) {
-		sc->sc_mbuf->m_pkthdr.len = sc->sc_mbuf->m_len +=
sizeof(*bus);
-		bus = sc->sc_statep.b;
-		bus->creatorid = pf_status.hostid;
-		bus->status = status;
-		bus->endtime = htonl(time_uptime - sc->sc_ureq_received);
-		pfsync_sendout(sc);
+	CLR(pd->pd_st->state_flags, PFSTATE_ACK);
+	timeout_del(&pd->pd_tmo); /* bah */
+	if (drop)
+		m_freem(pd->pd_m);
+	else {
+		s = splnet();
+		ip_output(pd->pd_m, (void *)NULL, (void *)NULL, 0,
+		    (void *)NULL, (void *)NULL);
+		splx(s);
 	}
+
+	pool_put(&sc->sc_pool, pd);
 }

 void
-pfsync_bulk_update(void *v)
+pfsync_defer_tmo(void *arg)
 {
-	struct pfsync_softc *sc = v;
-	int s, i = 0;
-	struct pf_state *state;
+	int s;

-	s = splnet();
-	if (sc->sc_mbuf != NULL)
-		pfsync_sendout(sc);
+	s = splsoftnet();
+	pfsync_undefer(arg, 0);
+	splx(s);
+}

-	/*
-	 * Grab at most PFSYNC_BULKPACKETS worth of states which have not
```

```
-	 * been sent since the latest request was made.
-	 */
-	state = sc->sc_bulk_send_next;
-	if (state)
-		do {
-			/* send state update if syncable and not already sent */
-			if (!state->sync_flags
-			    && state->timeout < PFTM_MAX
-			    && state->pfsync_time <= sc->sc_ureq_received) {
-				pfsync_pack_state(PFSYNC_ACT_UPD, state, 0);
-				i++;
-			}
+void
+pfsync_deferred(struct pf_state *st, int drop)
+{
+	struct pfsync_softc *sc = pfsyncif;
+	struct pfsync_deferral *pd;

-			/* figure next state to send */
-			state = TAILQ_NEXT(state, entry_list);
+	TAILQ_FOREACH(pd, &sc->sc_deferrals, pd_entry) {
+		 if (pd->pd_st == st) {
+			pfsync_undefer(pd, drop);
+			return;
+		}
+	}

-			/* wrap to start of list if we hit the end */
-			if (!state)
-				state = TAILQ_FIRST(&state_list);
-		} while (i < sc->sc_maxcount * PFSYNC_BULKPACKETS &&
-		    state != sc->sc_bulk_terminator);
-
-	if (!state || state == sc->sc_bulk_terminator) {
-		/* we're done */
-		pfsync_send_bus(sc, PFSYNC_BUS_END);
-		sc->sc_ureq_received = 0;
-		sc->sc_bulk_send_next = NULL;
-		sc->sc_bulk_terminator = NULL;
-		timeout_del(&sc->sc_bulk_tmo);
-		if (pf_status.debug >= PF_DEBUG_MISC)
-			printf("pfsync: bulk update complete\n");
-	} else {
-		/* look again for more in a bit */
-		timeout_add(&sc->sc_bulk_tmo, 1);
-		sc->sc_bulk_send_next = state;
+	panic("pfsync_send_deferred: unable to find deferred state");
+}
+
+u_int pfsync_upds = 0;
+
+void
+pfsync_update_state(struct pf_state *st)
+{
+	struct pfsync_softc *sc = pfsyncif;
+	int sync = 0;
```

```
+
+	splassert(IPL_SOFTNET);
+
+	if (sc == NULL)
+		return;
+
+	if (ISSET(st->state_flags, PFSTATE_ACK))
+		pfsync_deferred(st, 0);
+	if (ISSET(st->state_flags, PFSTATE_NOSYNC)) {
+		if (st->sync_state != PFSYNC_S_NONE)
+			pfsync_q_del(st);
+		return;
+	}
+
+	if (sc->sc_len == PFSYNC_MINPKT)
+		timeout_add_sec(&sc->sc_tmo, 1);
+
+	switch (st->sync_state) {
+	case PFSYNC_S_UPD_C:
+	case PFSYNC_S_UPD:
+	case PFSYNC_S_INS:
+		/* we're already handling it */
+
+		st->sync_updates++;
+		if (st->sync_updates >= sc->sc_maxupdates)
+			sync = 1;
+		break;
+
+	case PFSYNC_S_IACK:
+		pfsync_q_del(st);
+	case PFSYNC_S_NONE:
+		pfsync_q_ins(st, PFSYNC_S_UPD_C);
+		st->sync_updates = 0;
+		break;
+
+	default:
+		panic("pfsync_update_state: unexpected sync state %d",
+		    st->sync_state);
+	}
+
+	if (sync || (time_second - st->pfsync_time) < 2) {
+		pfsync_upds++;
+		schednetisr(NETISR_PFSYNC);
 	}
-	if (sc->sc_mbuf != NULL)
-		pfsync_sendout(sc);
-	splx(s);
 }

 void
-pfsync_bulkfail(void *v)
+pfsync_request_update(u_int32_t creatorid, u_int64_t id)
 {
-	struct pfsync_softc *sc = v;
-	int s, error;
+	struct pfsync_softc *sc = pfsyncif;
```

```
+	struct pfsync_upd_req_item *item;
+	size_t nlen = sizeof(struct pfsync_upd_req);
+	int s;

-	if (sc->sc_bulk_tries++ < PFSYNC_MAX_BULKTRIES) {
-		/* Try again in a bit */
-		timeout_add_sec(&sc->sc_bulkfail_tmo, 5);
+	/*
+	 * this code does nothing to prevent multiple update requests for
the
+	 * same state being generated.
+	 */
+
+	item = pool_get(&sc->sc_pool, PR_NOWAIT);
+	if (item == NULL) {
+		/* XXX stats */
+		return;
+	}
+
+	item->ur_msg.id = id;
+	item->ur_msg.creatorid = creatorid;
+
+	if (TAILQ_EMPTY(&sc->sc_upd_req_list))
+		nlen += sizeof(struct pfsync_subheader);
+
+	if (sc->sc_len + nlen > sc->sc_if.if_mtu) {
		s = splnet();
-		error = pfsync_request_update(NULL, NULL);
-		if (error == ENOMEM) {
-			if (pf_status.debug >= PF_DEBUG_MISC)
-				printf("pfsync: cannot allocate mbufs for "
-				    "bulk update\n");
-		} else
-			pfsync_sendout(sc);
+		pfsync_sendout();
		splx(s);
-	} else {
-		/* Pretend like the transfer was ok */
-		sc->sc_ureq_sent = 0;
-		sc->sc_bulk_tries = 0;
-#if NCARP > 0
-		if (!pfsync_sync_ok)
-			carp_group_demote_adj(&sc->sc_if, -1);
-#endif
-		pfsync_sync_ok = 1;
-		if (pf_status.debug >= PF_DEBUG_MISC)
-			printf("pfsync: failed to receive "
-			    "bulk update status\n");
-		timeout_del(&sc->sc_bulkfail_tmo);
+
+		nlen = sizeof(struct pfsync_subheader) +
+		    sizeof(struct pfsync_upd_req);
	}
+
+	TAILQ_INSERT_TAIL(&sc->sc_upd_req_list, item, ur_entry);
+	sc->sc_len += nlen;
```

```
+
+	schednetisr(NETISR_PFSYNC);
 }

-/* This must be called in splnet() */
-int
-pfsync_sendout(struct pfsync_softc *sc)
+void
+pfsync_update_state_req(struct pf_state *st)
 {
-#if NBPFILTER > 0
-	struct ifnet *ifp = &sc->sc_if;
-#endif
-	struct mbuf *m;
+	struct pfsync_softc *sc = pfsyncif;

-	timeout_del(&sc->sc_tmo);
+	if (sc == NULL)
+		panic("pfsync_update_state_req: nonexistant instance");

-	if (sc->sc_mbuf == NULL)
-		return (0);
-	m = sc->sc_mbuf;
-	sc->sc_mbuf = NULL;
-	sc->sc_statep.s = NULL;
+	if (ISSET(st->state_flags, PFSTATE_NOSYNC)) {
+		if (st->sync_state != PFSYNC_S_NONE)
+			pfsync_q_del(st);
+		return;
+	}

-#if NBPFILTER > 0
-	if (ifp->if_bpf)
-		bpf_mtap(ifp->if_bpf, m, BPF_DIRECTION_OUT);
-#endif
+	switch (st->sync_state) {
+	case PFSYNC_S_UPD_C:
+	case PFSYNC_S_IACK:
+		pfsync_q_del(st);
+	case PFSYNC_S_NONE:
+		pfsync_q_ins(st, PFSYNC_S_UPD);
+		schednetisr(NETISR_PFSYNC);
+		return;

-	if (sc->sc_mbuf_net) {
-		m_freem(m);
-		m = sc->sc_mbuf_net;
-		sc->sc_mbuf_net = NULL;
-		sc->sc_statep_net.s = NULL;
-	}
+	case PFSYNC_S_INS:
+	case PFSYNC_S_UPD:
+	case PFSYNC_S_DEL:
+		/* we're already handling it */
+		return;
```

```
-	return pfsync_sendout_mbuf(sc, m);
+	default:
+		panic("pfsync_update_state_req: unexpected sync state %d",
+		    st->sync_state);
+	}
 }

-int
-pfsync_tdb_sendout(struct pfsync_softc *sc)
+void
+pfsync_delete_state(struct pf_state *st)
 {
-#if NBPFILTER > 0
-	struct ifnet *ifp = &sc->sc_if;
-#endif
-	struct mbuf *m;
+	struct pfsync_softc *sc = pfsyncif;

-	timeout_del(&sc->sc_tdb_tmo);
+	splassert(IPL_SOFTNET);

-	if (sc->sc_mbuf_tdb == NULL)
-		return (0);
-	m = sc->sc_mbuf_tdb;
-	sc->sc_mbuf_tdb = NULL;
-	sc->sc_statep_tdb.t = NULL;
+	if (sc == NULL)
+		return;

-#if NBPFILTER > 0
-	if (ifp->if_bpf)
-		bpf_mtap(ifp->if_bpf, m, BPF_DIRECTION_OUT);
-#endif
+	if (ISSET(st->state_flags, PFSTATE_ACK))
+		pfsync_deferred(st, 1);
+	if (ISSET(st->state_flags, PFSTATE_NOSYNC)) {
+		if (st->sync_state != PFSYNC_S_NONE)
+			pfsync_q_del(st);
+		return;
+	}
+
+	if (sc->sc_len == PFSYNC_MINPKT)
+		timeout_add_sec(&sc->sc_tmo, 1);
+
+	switch (st->sync_state) {
+	case PFSYNC_S_INS:
+		/* we never got to tell the world so just forget about it */
+		pfsync_q_del(st);
+		return;

-	return pfsync_sendout_mbuf(sc, m);
+	case PFSYNC_S_UPD_C:
+	case PFSYNC_S_UPD:
+	case PFSYNC_S_IACK:
+		pfsync_q_del(st);
+		/* FALLTHROUGH to putting it on the del list */
```

```
+
+	case PFSYNC_S_NONE:
+		pfsync_q_ins(st, PFSYNC_S_DEL);
+		return;
+
+	default:
+		panic("pfsync_delete_state: unexpected sync state %d",
+		    st->sync_state);
+	}
 }

-int
-pfsync_sendout_mbuf(struct pfsync_softc *sc, struct mbuf *m)
+void
+pfsync_clear_states(u_int32_t creatorid, const char *ifname)
 {
-	struct sockaddr sa;
-	struct ip *ip;
+	struct {
+		struct pfsync_subheader subh;
+		struct pfsync_clr clr;
+	} __packed r;

-	if (sc->sc_sync_ifp ||
-	    sc->sc_sync_peer.s_addr != INADDR_PFSYNC_GROUP) {
-		M_PREPEND(m, sizeof(struct ip), M_DONTWAIT);
-		if (m == NULL) {
-			pfsyncstats.pfsyncs_onomem++;
-			return (0);
-		}
-		ip = mtod(m, struct ip *);
-		ip->ip_v = IPVERSION;
-		ip->ip_hl = sizeof(*ip) >> 2;
-		ip->ip_tos = IPTOS_LOWDELAY;
-		ip->ip_len = htons(m->m_pkthdr.len);
-		ip->ip_id = htons(ip_randomid());
-		ip->ip_off = htons(IP_DF);
-		ip->ip_ttl = PFSYNC_DFLTTL;
-		ip->ip_p = IPPROTO_PFSYNC;
-		ip->ip_sum = 0;
+	struct pfsync_softc *sc = pfsyncif;

-		bzero(&sa, sizeof(sa));
-		ip->ip_src.s_addr = INADDR_ANY;
+	splassert(IPL_SOFTNET);
+
+	if (sc == NULL)
+		return;

-		if (sc->sc_sendaddr.s_addr == INADDR_PFSYNC_GROUP)
-			m->m_flags |= M_MCAST;
-		ip->ip_dst = sc->sc_sendaddr;
-		sc->sc_sendaddr.s_addr = sc->sc_sync_peer.s_addr;
+	bzero(&r, sizeof(r));

-		pfsyncstats.pfsyncs_opackets++;
```

```
+	r.subh.action = PFSYNC_ACT_CLR;
+	r.subh.count = htons(1);

-		if (ip_output(m, NULL, NULL, IP_RAWOUTPUT, &sc->sc_imo, NULL))
-			pfsyncstats.pfsyncs_oerrors++;
-	} else
-		m_freem(m);
+	strlcpy(r.clr.ifname, ifname, sizeof(r.clr.ifname));
+	r.clr.creatorid = creatorid;

-	return (0);
+	pfsync_send_plus(&r, sizeof(r));
 }

-#ifdef IPSEC
-/* Update an in-kernel tdb. Silently fail if no tdb is found. */
 void
-pfsync_update_net_tdb(struct pfsync_tdb *pt)
+pfsync_q_ins(struct pf_state *st, int q)
 {
-	struct tdb		*tdb;
-	int			 s;
+	struct pfsync_softc *sc = pfsyncif;
+	size_t nlen = pfsync_qs[q].len;
+	int s;

-	/* check for invalid values */
-	if (ntohl(pt->spi) <= SPI_RESERVED_MAX ||
-	    (pt->dst.sa.sa_family != AF_INET &&
-	     pt->dst.sa.sa_family != AF_INET6))
-		goto bad;
-
-	s = spltdb();
-	tdb = gettdb(pt->spi, &pt->dst, pt->sproto);
-	if (tdb) {
-		pt->rpl = ntohl(pt->rpl);
-		pt->cur_bytes = betoh64(pt->cur_bytes);
-
-		/* Neither replay nor byte counter should ever decrease. */
-		if (pt->rpl < tdb->tdb_rpl ||
-		    pt->cur_bytes < tdb->tdb_cur_bytes) {
-			splx(s);
-			goto bad;
-		}
+	KASSERT(st->sync_state == PFSYNC_S_NONE);
+
+#if 1 || defined(PFSYNC_DEBUG)
+	if (sc->sc_len < PFSYNC_MINPKT)
+		panic("pfsync pkt len is too low %d", sc->sc_len);
+#endif
+	if (TAILQ_EMPTY(&sc->sc_qs[q]))
+		nlen += sizeof(struct pfsync_subheader);

-		tdb->tdb_rpl = pt->rpl;
-		tdb->tdb_cur_bytes = pt->cur_bytes;
+	if (sc->sc_len + nlen > sc->sc_if.if_mtu) {
```

```
+           s = splnet();
+           pfsync_sendout();
+           splx(s);
+
+           nlen = sizeof(struct pfsync_subheader) + pfsync_qs[q].len;
        }
-       splx(s);
-       return;

- bad:
-       if (pf_status.debug >= PF_DEBUG_MISC)
-           printf("pfsync_insert: PFSYNC_ACT_TDB_UPD: "
-               "invalid value\n");
-       pfsyncstats.pfsyncs_badstate++;
-       return;
+       sc->sc_len += nlen;
+       TAILQ_INSERT_TAIL(&sc->sc_qs[q], st, sync_list);
+       st->sync_state = q;
 }

-/* One of our local tdbs have been updated, need to sync rpl with
others */
-int
-pfsync_update_tdb(struct tdb *tdb, int output)
+void
+pfsync_q_del(struct pf_state *st)
 {
-       struct ifnet *ifp = NULL;
        struct pfsync_softc *sc = pfsyncif;
-       struct pfsync_header *h;
-       struct pfsync_tdb *pt = NULL;
-       int s, i, ret;
+       int q = st->sync_state;

-       if (sc == NULL)
-           return (0);
+       KASSERT(st->sync_state != PFSYNC_S_NONE);

-       ifp = &sc->sc_if;
-       if (ifp->if_bpf == NULL && sc->sc_sync_ifp == NULL &&
-           sc->sc_sync_peer.s_addr == INADDR_PFSYNC_GROUP) {
-           /* Don't leave any stale pfsync packets hanging around. */
-           if (sc->sc_mbuf_tdb != NULL) {
-               m_freem(sc->sc_mbuf_tdb);
-               sc->sc_mbuf_tdb = NULL;
-               sc->sc_statep_tdb.t = NULL;
-           }
-           return (0);
-       }
+       sc->sc_len -= pfsync_qs[q].len;
+       TAILQ_REMOVE(&sc->sc_qs[q], st, sync_list);
+       st->sync_state = PFSYNC_S_NONE;

-       s = splnet();
-       if (sc->sc_mbuf_tdb == NULL) {
-           if ((sc->sc_mbuf_tdb = pfsync_get_mbuf(sc, PFSYNC_ACT_TDB_UPD,
-               (void *)&sc->sc_statep_tdb.t)) == NULL) {
+       if (TAILQ_EMPTY(&sc->sc_qs[q]))
+               sc->sc_len -= sizeof(struct pfsync_subheader);
+}
+
+void
+pfsync_update_tdb(struct tdb *t, int output)
+{
+       struct pfsync_softc *sc = pfsyncif;
+       size_t nlen = sizeof(struct pfsync_tdb);
+       int s;
+
+       if (sc == NULL)
+               return;
+
+       if (!ISSET(t->tdb_flags, TDBF_PFSYNC)) {
+               if (TAILQ_EMPTY(&sc->sc_tdb_q))
+                       nlen += sizeof(struct pfsync_subheader);
+
+               if (sc->sc_len + nlen > sc->sc_if.if_mtu) {
+                       s = splnet();
+                       pfsync_sendout();
                        splx(s);
-                       return (ENOMEM);
-               }
-               h = mtod(sc->sc_mbuf_tdb, struct pfsync_header *);
-       } else {
-               h = mtod(sc->sc_mbuf_tdb, struct pfsync_header *);
-               if (h->action != PFSYNC_ACT_TDB_UPD) {
-                       /*
-                        * XXX will never happen as long as there's
-                        * only one "TDB action".
-                        */
-                       pfsync_tdb_sendout(sc);
-                       sc->sc_mbuf_tdb = pfsync_get_mbuf(sc,
-                           PFSYNC_ACT_TDB_UPD, (void *)&sc->sc_statep_tdb.t);
-                       if (sc->sc_mbuf_tdb == NULL) {
-                               splx(s);
-                               return (ENOMEM);
-                       }
-                       h = mtod(sc->sc_mbuf_tdb, struct pfsync_header *);
-               } else if (sc->sc_maxupdates) {
-                       /*
-                        * If it's an update, look in the packet to see if
-                        * we already have an update for the state.
-                        */
-                       struct pfsync_tdb *u =
-                           (void *)((char *)h + PFSYNC_HDRLEN);

-                       for (i = 0; !pt && i < h->count; i++) {
-                               if (tdb->tdb_spi == u->spi &&
-                                   tdb->tdb_sproto == u->sproto &&
-                                   !bcmp(&tdb->tdb_dst, &u->dst,
-                                   SA_LEN(&u->dst.sa))) {
-                                     pt = u;
-                                     pt->updates++;
-                       }
-                       u++;
-                 }
+                 nlen = sizeof(struct pfsync_subheader) +
+                     sizeof(struct pfsync_tdb);
            }
-     }

-     if (pt == NULL) {
-           /* not a "duplicate" update */
-           pt = sc->sc_statep_tdb.t++;
-           sc->sc_mbuf_tdb->m_pkthdr.len =
-               sc->sc_mbuf_tdb->m_len += sizeof(struct pfsync_tdb);
-           h->count++;
-           bzero(pt, sizeof(*pt));
-
-           pt->spi = tdb->tdb_spi;
-           memcpy(&pt->dst, &tdb->tdb_dst, sizeof pt->dst);
-           pt->sproto = tdb->tdb_sproto;
+           sc->sc_len += nlen;
+           TAILQ_INSERT_TAIL(&sc->sc_tdb_q, t, tdb_sync_entry);
+           SET(t->tdb_flags, TDBF_PFSYNC);
+           t->tdb_updates = 0;
+     } else {
+           if (++t->tdb_updates >= sc->sc_maxupdates)
+                 schednetisr(NETISR_PFSYNC);
      }

+     if (output)
+           SET(t->tdb_flags, TDBF_PFSYNC_RPL);
+     else
+           CLR(t->tdb_flags, TDBF_PFSYNC_RPL);
+}
+
+void
+pfsync_delete_tdb(struct tdb *t)
+{
+     struct pfsync_softc *sc = pfsyncif;
+
+     if (sc == NULL || !ISSET(t->tdb_flags, TDBF_PFSYNC))
+           return;
+
+     sc->sc_len -= sizeof(struct pfsync_tdb);
+     TAILQ_REMOVE(&sc->sc_tdb_q, t, tdb_sync_entry);
+     CLR(t->tdb_flags, TDBF_PFSYNC);
+
+     if (TAILQ_EMPTY(&sc->sc_tdb_q))
+           sc->sc_len -= sizeof(struct pfsync_subheader);
+}
+
+int
+pfsync_out_tdb(struct tdb *t, struct mbuf *m, int offset)
+{
+     struct pfsync_tdb *ut = (struct pfsync_tdb *)(m->m_data + offset);
+
+     bzero(ut, sizeof(*ut));
```

```
+	ut->spi = t->tdb_spi;
+	bcopy(&t->tdb_dst, &ut->dst, sizeof(ut->dst));
 	/*
 	 * When a failover happens, the master's rpl is probably above
 	 * what we see here (we may be up to a second late), so
@@ -1984,17 +2158,184 @@ pfsync_update_tdb(struct tdb *tdb, int o
 	 * this edge case.
 	 */
 #define RPL_INCR 16384
-	pt->rpl = htonl(tdb->tdb_rpl + (output ? RPL_INCR : 0));
-	pt->cur_bytes = htobe64(tdb->tdb_cur_bytes);
+	ut->rpl = htonl(t->tdb_rpl + (ISSET(t->tdb_flags,
TDBF_PFSYNC_RPL) ?
+	    RPL_INCR : 0));
+	ut->cur_bytes = htobe64(t->tdb_cur_bytes);
+	ut->sproto = t->tdb_sproto;

-	if (h->count == sc->sc_maxcount ||
-	    (sc->sc_maxupdates && (pt->updates >= sc->sc_maxupdates)))
-		ret = pfsync_tdb_sendout(sc);
+	return (sizeof(*ut));
+}

-	splx(s);
-	return (ret);
+void
+pfsync_bulk_start(void)
+{
+	struct pfsync_softc *sc = pfsyncif;
+
+	sc->sc_ureq_received = time_uptime;
+
+	if (sc->sc_bulk_next == NULL)
+		sc->sc_bulk_next = TAILQ_FIRST(&state_list);
+	sc->sc_bulk_last = sc->sc_bulk_next;
+
+	if (pf_status.debug >= PF_DEBUG_MISC)
+		printf("pfsync: received bulk update request\n");
+
+	pfsync_bulk_status(PFSYNC_BUS_START);
+	pfsync_bulk_update(sc);
+}
+
+void
+pfsync_bulk_update(void *arg)
+{
+	struct pfsync_softc *sc = arg;
+	struct pf_state *st = sc->sc_bulk_next;
+	int i = 0;
+
+	do {
+		if (st->sync_state == PFSYNC_S_NONE &&
+		    st->timeout < PFTM_MAX &&
+		    st->pfsync_time <= sc->sc_ureq_received) {
+			pfsync_update_state_req(st);
+			i++;
+		}
+
+		st = TAILQ_NEXT(st, entry_list);
+		if (st == NULL)
+			st = TAILQ_FIRST(&state_list);
+
+		if (i > 0 && TAILQ_EMPTY(&sc->sc_qs[PFSYNC_S_UPD])) {
+			sc->sc_bulk_next = st;
+			timeout_add(&sc->sc_bulk_tmo, 1);
+			return;
+		}
+	} while (st != sc->sc_bulk_last);
+
+	/* we're done */
+	sc->sc_bulk_next = NULL;
+	sc->sc_bulk_last = NULL;
+	pfsync_bulk_status(PFSYNC_BUS_END);
 }
+
+void
+pfsync_bulk_status(u_int8_t status)
+{
+	struct {
+		struct pfsync_subheader subh;
+		struct pfsync_bus bus;
+	} __packed r;
+
+	struct pfsync_softc *sc = pfsyncif;
+
+	bzero(&r, sizeof(r));
+
+	r.subh.action = PFSYNC_ACT_BUS;
+	r.subh.count = htons(1);
+
+	r.bus.creatorid = pf_status.hostid;
+	r.bus.endtime = htonl(time_uptime - sc->sc_ureq_received);
+	r.bus.status = status;
+
+	pfsync_send_plus(&r, sizeof(r));
+}
+
+void
+pfsync_bulk_fail(void *arg)
+{
+	struct pfsync_softc *sc = arg;
+
+	if (sc->sc_bulk_tries++ < PFSYNC_MAX_BULKTRIES) {
+		/* Try again */
+		timeout_add_sec(&sc->sc_bulkfail_tmo, 5);
+		pfsync_request_update(0, 0);
+	} else {
+		/* Pretend like the transfer was ok */
+		sc->sc_ureq_sent = 0;
+		sc->sc_bulk_tries = 0;
+#if NCARP > 0
+		if (!pfsync_sync_ok)
```

```
+               carp_group_demote_adj(&sc->sc_if, -1);
 #endif
+          pfsync_sync_ok = 1;
+          if (pf_status.debug >= PF_DEBUG_MISC)
+               printf("pfsync: failed to receive bulk update\n");
+     }
+}
+
+void
+pfsync_send_plus(void *plus, size_t pluslen)
+{
+     struct pfsync_softc *sc = pfsyncif;
+     int s;
+
+     if (sc->sc_len + pluslen > sc->sc_if.if_mtu) {
+          s = splnet();
+          pfsync_sendout();
+          splx(s);
+     }
+
+     sc->sc_plus = plus;
+     sc->sc_len += (sc->sc_pluslen = pluslen);
+
+     s = splnet();
+     pfsync_sendout();
+     splx(s);
+}
+
+int
+pfsync_up(void)
+{
+     struct pfsync_softc *sc = pfsyncif;
+
+     if (sc == NULL || !ISSET(sc->sc_if.if_flags, IFF_RUNNING))
+          return (0);
+
+     return (1);
+}
+
+int
+pfsync_state_in_use(struct pf_state *st)
+{
+     struct pfsync_softc *sc = pfsyncif;
+
+     if (sc == NULL)
+          return (0);
+
+     if (st->sync_state != PFSYNC_S_NONE)
+          return (1);
+
+     if (sc->sc_bulk_next == NULL && sc->sc_bulk_last == NULL)
+          return (0);
+
+     return (1);
+}
+
+u_int pfsync_ints;
+u_int pfsync_tmos;
+
+void
+pfsync_timeout(void *arg)
+{
+	int s;
+
+	pfsync_tmos++;
+
+	s = splnet();
+	pfsync_sendout();
+	splx(s);
+}
+
+/* this is a softnet/netisr handler */
+void
+pfsyncintr(void)
+{
+	int s;
+
+	pfsync_ints++;
+
+	s = splnet();
+	pfsync_sendout();
+	splx(s);
+}

 int
 pfsync_sysctl(int *name, u_int namelen, void *oldp, size_t *oldlenp,
void *newp,
Index: sys/net/if_pfsync.h
===================================================================
RCS file: /cvs/src/sys/net/if_pfsync.h,v
retrieving revision 1.35
diff -u -p -r1.35 if_pfsync.h
--- sys/net/if_pfsync.h	29 Jun 2008 08:42:15 -0000	1.35
+++ sys/net/if_pfsync.h	4 Feb 2009 05:18:14 -0000
@@ -26,154 +26,217 @@
  * THE POSSIBILITY OF SUCH DAMAGE.
  */

+/*
+ * Copyright (c) 2008 David Gwynne <dlg@openbsd.org>
+ *
+ * Permission to use, copy, modify, and distribute this software for
any
+ * purpose with or without fee is hereby granted, provided that the
above
+ * copyright notice and this permission notice appear in all copies.
+ *
+ * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL
WARRANTIES
+ * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
+ * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE
FOR
```

```
+ * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY
DAMAGES
+ * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN
AN
+ * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT
OF
+ * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
+ */
+
 #ifndef _NET_IF_PFSYNC_H_
 #define _NET_IF_PFSYNC_H_

+#define PFSYNC_VERSION         5
+#define PFSYNC_DFLTTL          255

-#define PFSYNC_ID_LEN     sizeof(u_int64_t)
+#define PFSYNC_ACT_CLR         0     /* clear all states */
+#define PFSYNC_ACT_INS         1     /* insert state */
+#define PFSYNC_ACT_INS_ACK     2     /* ack of insterted state */
+#define PFSYNC_ACT_UPD         3     /* update state */
+#define PFSYNC_ACT_UPD_C  4     /* "compressed" update state */
+#define PFSYNC_ACT_UPD_REQ     5     /* request "uncompressed" state */
+#define PFSYNC_ACT_DEL         6     /* delete state */
+#define PFSYNC_ACT_DEL_C  7     /* "compressed" delete state */
+#define PFSYNC_ACT_INS_F  8     /* insert fragment */
+#define PFSYNC_ACT_DEL_F  9     /* delete fragments */
+#define PFSYNC_ACT_BUS         10    /* bulk update status */
+#define PFSYNC_ACT_TDB         11    /* TDB replay counter update */
+#define PFSYNC_ACT_EOF         12    /* end of frame */
+#define PFSYNC_ACT_MAX         13
+
+#define PFSYNC_ACTIONS         "CLR ST",       \
+                     "INS ST",       \
+                     "INS ST ACK",        \
+                     "UPD ST",       \
+                     "UPD ST COMP",       \
+                     "UPD ST REQ",        \
+                     "DEL ST",       \
+                     "DEL ST COMP",       \
+                     "INS FR",       \
+                     "DEL FR",       \
+                     "BULK UPD STAT",\
+                     "TDB UPD",      \
+                     "EOF"

-struct pfsync_tdb {
-     u_int32_t  spi;
-     union sockaddr_union dst;
-     u_int32_t  rpl;
-     u_int64_t  cur_bytes;
-     u_int8_t   sproto;
-     u_int8_t   updates;
-     u_int8_t   pad[2];
+#define PFSYNC_HMAC_LEN   20
+
+/*
```

```
+ * A pfsync frame is built from a header followed by several sections
which
+ * are all prefixed with their own subheaders. Frames must be
terminated with
+ * an EOF subheader.
+ *
+ * | ...                |
+ * | IP header              |
+ * +============================+
+ * | pfsync_header          |
+ * +----------------------------+
+ * | pfsync_subheader           |
+ * +----------------------------+
+ * | first action fields  |
+ * | ...                |
+ * +----------------------------+
+ * | pfsync_subheader           |
+ * +----------------------------+
+ * | second action fields |
+ * | ...                |
+ * +----------------------------+
+ * | EOF pfsync_subheader |
+ * +----------------------------+
+ * | HMAC               |
+ * +============================+
+ */
+
+/*
+ * Frame header
+ */
+
+struct pfsync_header {
+	u_int8_t		version;
+	u_int8_t		_pad;
+	u_int16_t		len;
+	u_int8_t		pfcksum[PF_MD5_DIGEST_LENGTH];
+} __packed;
+
+/*
+ * Frame region subheader
+ */
+
+struct pfsync_subheader {
+	u_int8_t		action;
+	u_int8_t		_pad;
+	u_int16_t		count;
+} __packed;
+
+/*
+ * CLR
+ */
+
+struct pfsync_clr {
+	char			ifname[IFNAMSIZ];
+	u_int32_t		creatorid;
 } __packed;
```

```
-struct pfsync_state_upd {
-	u_int32_t	id[2];
+/*
+ * INS, UPD, DEL
+ */
+
+/* these use struct pfsync_state in pfvar.h */
+
+/*
+ * INS_ACK
+ */
+
+struct pfsync_ins_ack {
+	u_int64_t		id;
+	u_int32_t		creatorid;
+} __packed;
+
+/*
+ * UPD_C
+ */
+
+struct pfsync_upd_c {
+	u_int64_t		id;
 	struct pfsync_state_peer	src;
 	struct pfsync_state_peer	dst;
-	u_int32_t	creatorid;
-	u_int32_t	expire;
-	u_int8_t	timeout;
-	u_int8_t	updates;
-	u_int8_t	pad[6];
-} __packed;
-
-struct pfsync_state_del {
-	u_int32_t	id[2];
-	u_int32_t	creatorid;
-	struct {
-		u_int8_t	state;
-	} src;
-	struct {
-		u_int8_t	state;
-	} dst;
-	u_int8_t	pad[2];
-} __packed;
-
-struct pfsync_state_upd_req {
-	u_int32_t	id[2];
-	u_int32_t	creatorid;
-	u_int32_t	pad;
-} __packed;
-
-struct pfsync_state_clr {
-	char		ifname[IFNAMSIZ];
-	u_int32_t	creatorid;
-	u_int32_t	pad;
-} __packed;
```

```
-
-struct pfsync_state_bus {
-	u_int32_t	creatorid;
-	u_int32_t	endtime;
-	u_int8_t	status;
-#define PFSYNC_BUS_START	1
-#define PFSYNC_BUS_END		2
-	u_int8_t	pad[7];
+	u_int32_t		creatorid;
+	u_int32_t		expire;
+	u_int8_t		timeout;
+	u_int8_t		_pad[3];
 } __packed;

 /*
- * Names for PFSYNC sysctl objects
+ * UPD_REQ
  */
-#define	PFSYNCCTL_STATS		1	/* PFSYNC stats */
-#define	PFSYNCCTL_MAXID		2

-#define	PFSYNCCTL_NAMES { \
-	{ 0, 0 }, \
-	{ "stats", CTLTYPE_STRUCT }, \
-}
+struct pfsync_upd_req {
+	u_int64_t		id;
+	u_int32_t		creatorid;
+} __packed;

-#ifdef _KERNEL
+/*
+ * DEL_C
+ */

-union sc_statep {
-	struct pfsync_state	*s;
-	struct pfsync_state_upd	*u;
-	struct pfsync_state_del	*d;
-	struct pfsync_state_clr	*c;
-	struct pfsync_state_bus	*b;
-	struct pfsync_state_upd_req	*r;
-};
+struct pfsync_del_c {
+	u_int64_t		id;
+	u_int32_t		creatorid;
+} __packed;

-union sc_tdb_statep {
-	struct pfsync_tdb	*t;
-};
+/*
+ * INS_F, DEL_F
+ */

-extern int	pfsync_sync_ok;
```

```
+/* not implemented (yet) */

-struct pfsync_softc {
-     struct ifnet           sc_if;
-     struct ifnet          *sc_sync_ifp;
-
-     struct ip_moptions     sc_imo;
-     struct timeout         sc_tmo;
-     struct timeout         sc_tdb_tmo;
-     struct timeout         sc_bulk_tmo;
-     struct timeout         sc_bulkfail_tmo;
-     struct in_addr         sc_sync_peer;
-     struct in_addr         sc_sendaddr;
-     struct mbuf           *sc_mbuf;  /* current cumulative mbuf */
-     struct mbuf           *sc_mbuf_net;   /* current cumulative mbuf */
-     struct mbuf           *sc_mbuf_tdb;   /* dito for TDB updates */
-     union sc_statep        sc_statep;
-     union sc_statep        sc_statep_net;
-     union sc_tdb_statep    sc_statep_tdb;
-     u_int32_t         sc_ureq_received;
-     u_int32_t         sc_ureq_sent;
-     struct pf_state       *sc_bulk_send_next;
-     struct pf_state       *sc_bulk_terminator;
-     int               sc_bulk_tries;
-     int               sc_maxcount;   /* number of states in mtu */
-     int               sc_maxupdates; /* number of updates/state */
-};
+/*
+ * BUS
+ */

-extern struct pfsync_softc     *pfsyncif;
-#endif
+struct pfsync_bus {
+     u_int32_t             creatorid;
+     u_int32_t             endtime;
+     u_int8_t              status;
+#define PFSYNC_BUS_START            1
+#define PFSYNC_BUS_END                    2
+     u_int8_t              _pad[3];
+} __packed;

+/*
+ * TDB
+ */

-struct pfsync_header {
-     u_int8_t version;
-#define   PFSYNC_VERSION  4
-     u_int8_t af;
-     u_int8_t action;
-#define   PFSYNC_ACT_CLR       0     /* clear all states */
-#define   PFSYNC_ACT_INS       1     /* insert state */
-#define   PFSYNC_ACT_UPD       2     /* update state */
-#define   PFSYNC_ACT_DEL       3     /* delete state */
-#define   PFSYNC_ACT_UPD_C4    /* "compressed" state update */
```

```
-#define   PFSYNC_ACT_DEL_C5     /* "compressed" state delete */
-#define   PFSYNC_ACT_INS_F6     /* insert fragment */
-#define   PFSYNC_ACT_DEL_F7     /* delete fragments */
-#define   PFSYNC_ACT_UREQ       8     /* request "uncompressed" state */
-#define PFSYNC_ACT_BUS          9     /* Bulk Update Status */
-#define PFSYNC_ACT_TDB_UPD      10    /* TDB replay counter update */
-#define   PFSYNC_ACT_MAX        11
-     u_int8_t count;
-     u_int8_t pf_chksum[PF_MD5_DIGEST_LENGTH];
-} __packed;
-
-#define PFSYNC_BULKPACKETS      1     /* # of packets per timeout */
-#define PFSYNC_MAX_BULKTRIES    12
-#define PFSYNC_HDRLEN     sizeof(struct pfsync_header)
-#define   PFSYNC_ACTIONS \
-     "CLR ST", "INS ST", "UPD ST", "DEL ST", \
-     "UPD ST COMP", "DEL ST COMP", "INS FR", "DEL FR", \
-     "UPD REQ", "BLK UPD STAT", "TDB UPD"
+struct pfsync_tdb {
+     u_int32_t            spi;
+     union sockaddr_union      dst;
+     u_int32_t            rpl;
+     u_int64_t            cur_bytes;
+     u_int8_t             sproto;
+     u_int8_t             updates;
+     u_int8_t             _pad[2];
+} __packed;

-#define PFSYNC_DFLTTL           255
+/*
+ * EOF
+ */
+
+struct pfsync_eof {
+     u_int8_t             hmac[PFSYNC_HMAC_LEN];
+} __packed;
+
+#define PFSYNC_HDRLEN           sizeof(struct pfsync_header)
+
+
+
+/*
+ * Names for PFSYNC sysctl objects
+ */
+#define   PFSYNCCTL_STATS       1     /* PFSYNC stats */
+#define   PFSYNCCTL_MAXID       2
+
+#define   PFSYNCCTL_NAMES { \
+     { 0, 0 }, \
+     { "stats", CTLTYPE_STRUCT }, \
+}

 struct pfsyncstats {
     u_int64_t  pfsyncs_ipackets;    /* total input packets, IPv4 */
@@ -206,39 +269,43 @@ struct pfsyncreq {
 };
 #ifdef _KERNEL
+
+/*
+ * this shows where a pf state is with respect to the syncing.
+ */
+#define PFSYNC_S_INS 0x00
+#define PFSYNC_S_IACK     0x01
+#define PFSYNC_S_UPD 0x02
+#define PFSYNC_S_UPD_C    0x03
+#define PFSYNC_S_DEL 0x04
+#define PFSYNC_S_COUNT    0x05
+
+#define PFSYNC_S_DEFER    0xfe
+#define PFSYNC_S_NONE     0xff
+
 void           pfsync_input(struct mbuf *, ...);
-int            pfsync_clear_states(u_int32_t, char *);
-int            pfsync_pack_state(u_int8_t, struct pf_state *, int);
 int            pfsync_sysctl(int *, u_int,  void *, size_t *,
                    void *, size_t);
-void           pfsync_state_export(struct pfsync_state *,
-                   struct pf_state *);

 #define   PFSYNC_SI_IOCTL       0x01
 #define   PFSYNC_SI_CKSUM       0x02
+#define   PFSYNC_SI_ACK         0x04
 int            pfsync_state_import(struct pfsync_state *, u_int8_t);
+void           pfsync_state_export(struct pfsync_state *,
+                   struct pf_state *);
+
+void           pfsync_insert_state(struct pf_state *);
+void           pfsync_update_state(struct pf_state *);
+void           pfsync_delete_state(struct pf_state *);
+void           pfsync_clear_states(u_int32_t, const char *);
+
+void           pfsync_update_tdb(struct tdb *, int);
+void           pfsync_delete_tdb(struct tdb *);
+
+int            pfsync_defer(struct pf_state *, struct mbuf *);

-#define pfsync_insert_state(st)do {                     \
-    if ((st->rule.ptr->rule_flag & PFRULE_NOSYNC) ||\
-        (st->key[PF_SK_WIRE]->proto == IPPROTO_PFSYNC))   \
-          st->sync_flags |= PFSTATE_NOSYNC;              \
-    else if (!st->sync_flags)                      \
-          pfsync_pack_state(PFSYNC_ACT_INS, (st),    \
-              PFSYNC_FLAG_COMPRESS);               \
-    st->sync_flags &= ~PFSTATE_FROMSYNC;                \
-} while (0)
-#define pfsync_update_state(st) do {                    \
-    if (!st->sync_flags)                           \
-          pfsync_pack_state(PFSYNC_ACT_UPD, (st),    \
-              PFSYNC_FLAG_COMPRESS);               \
-    st->sync_flags &= ~PFSTATE_FROMSYNC;                \
-} while (0)
```

```
-#define pfsync_delete_state(st) do {                    \
-	if (!st->sync_flags)                            \
-		pfsync_pack_state(PFSYNC_ACT_DEL, (st),          \
-		    PFSYNC_FLAG_COMPRESS);              \
-} while (0)
-int pfsync_update_tdb(struct tdb *, int);
+int		pfsync_up(void);
+int		pfsync_state_in_use(struct pf_state *);
 #endif

 #endif /* _NET_IF_PFSYNC_H_ */
Index: sys/net/netisr.h
===================================================================
RCS file: /cvs/src/sys/net/netisr.h,v
retrieving revision 1.33
diff -u -p -r1.33 netisr.h
--- sys/net/netisr.h	9 May 2008 12:54:52 -0000	1.33
+++ sys/net/netisr.h	4 Feb 2009 05:18:14 -0000
@@ -56,6 +56,7 @@
 #define	NETISR_IP	2		/* same as AF_INET */
 #define	NETISR_TX	3		/* for if_snd processing */
 #define	NETISR_MPLS	4		/* AF_MPLS would overflow */
+#define NETISR_PFSYNC	5		/* for pfsync "immediate" tx */
 #define	NETISR_ATALK	16		/* same as AF_APPLETALK */
 #define	NETISR_ARP	18		/* same as AF_LINK */
 #define	NETISR_IPV6	24		/* same as AF_INET6 */
@@ -82,6 +83,7 @@ void	bridgeintr(void);
 void	pppoeintr(void);
 void	btintr(void);
 void	mplsintr(void);
+void	pfsyncintr(void);

 #include <machine/atomic.h>
 #define	schednetisr(anisr)					\
Index: sys/net/netisr_dispatch.h
===================================================================
RCS file: /cvs/src/sys/net/netisr_dispatch.h,v
retrieving revision 1.16
diff -u -p -r1.16 netisr_dispatch.h
--- sys/net/netisr_dispatch.h	7 May 2008 05:51:12 -0000	1.16
+++ sys/net/netisr_dispatch.h	4 Feb 2009 05:18:14 -0000
@@ -27,6 +27,7 @@
 #include "ppp.h"
 #include "bridge.h"
 #include "pppoe.h"
+#include "pfsync.h"
 #endif

 /*
@@ -63,5 +64,8 @@
 #endif
 #if NBLUETOOTH > 0
 	DONETISR(NETISR_BT,btintr);
+#endif
+#if NPFSYNC > 0
+	DONETISR(NETISR_PFSYNC,pfsyncintr);
```

```
 #endif
 	DONETISR(NETISR_TX,nettxintr);
Index: sys/net/pf.c
===================================================================
RCS file: /cvs/src/sys/net/pf.c,v
retrieving revision 1.632
diff -u -p -r1.632 pf.c
--- sys/net/pf.c30 Jan 2009 17:27:20 -0000 1.632
+++ sys/net/pf.c4 Feb 2009 05:18:14 -0000
@@ -801,8 +801,6 @@ pf_state_insert(struct pfi_kif *kif, str
 				printf("pf: state insert failed: "
 				    "id: %016llx creatorid: %08x",
 				    betoh64(s->id), ntohl(s->creatorid));
-				if (s->sync_flags & PFSTATE_FROMSYNC)
-					printf(" (from sync)");
 				printf("\n");
 			}
 			pf_detach_state(s);
@@ -1070,8 +1068,7 @@ pf_unlink_state(struct pf_state *cur)
 		export_pflow(cur);
 #endif
 #if NPFSYNC > 0
-	if (cur->creatorid == pf_status.hostid)
-		pfsync_delete_state(cur);
+	pfsync_delete_state(cur);
 #endif
 	cur->timeout = PFTM_UNLINKED;
 	pf_src_tree_remove_state(cur);
@@ -1086,9 +1083,7 @@ pf_free_state(struct pf_state *cur)
 	splassert(IPL_SOFTNET);
 
 #if NPFSYNC > 0
-	if (pfsyncif != NULL &&
-	    (pfsyncif->sc_bulk_send_next == cur ||
-	    pfsyncif->sc_bulk_terminator == cur))
+	if (pfsync_state_in_use(cur))
 		return;
 #endif
 	KASSERT(cur->timeout == PFTM_UNLINKED);
@@ -2813,6 +2808,20 @@ pf_test_rule(struct pf_rule **rm, struct
 	if (rewrite)
 		m_copyback(m, off, hdrlen, pd->hdr.any);
 
+#if NPFSYNC > 0
+	if (*sm != NULL && !ISSET((*sm)->state_flags, PFSTATE_NOSYNC) &&
+	    direction == PF_OUT && pfsync_up()) {
+		/*
+		 * We want the state created, but we dont
+		 * want to send this in case a partner
+		 * firewall has to know about it to allow
+		 * replies through it.
+		 */
+		if (pfsync_defer(*sm, m))
+			return (PF_DEFER);
+	}
+#endif
+
 	return (PF_PASS);

 cleanup:
@@ -2872,6 +2881,7 @@ pf_create_state(struct pf_rule *r, struc
 	if (r->rule_flag & PFRULE_PFLOW)
 		s->state_flags |= PFSTATE_PFLOW;
 	s->log = r->log & PF_LOG_ALL;
+	s->sync_state = PFSYNC_S_NONE;
 	if (nr != NULL)
 		s->log |= nr->log & PF_LOG_ALL;
 	switch (pd->proto) {
@@ -5335,14 +5345,19 @@ done:
 			    r->action == PF_PASS, tr->dst.neg);
 	}

-
-	if (action == PF_SYNPROXY_DROP) {
+	switch (action) {
+	case PF_SYNPROXY_DROP:
 		m_freem(*m0);
+	case PF_DEFER:
 		*m0 = NULL;
 		action = PF_PASS;
-	} else if (r->rt)
+		break;
+	default:
 		/* pf_route can free the mbuf causing *m0 to become NULL */
-		pf_route(m0, r, dir, kif->pfik_ifp, s, &pd);
+		if (r->rt)
+			pf_route(m0, r, dir, kif->pfik_ifp, s, &pd);
+		break;
+	}

 	return (action);
 }
@@ -5707,14 +5722,19 @@ done:
 			    r->action == PF_PASS, tr->dst.neg);
 	}

-
-	if (action == PF_SYNPROXY_DROP) {
+	switch (action) {
+	case PF_SYNPROXY_DROP:
 		m_freem(*m0);
+	case PF_DEFER:
 		*m0 = NULL;
 		action = PF_PASS;
-	} else if (r->rt)
+		break;
+	default:
 		/* pf_route6 can free the mbuf causing *m0 to become NULL */
-		pf_route6(m0, r, dir, kif->pfik_ifp, s, &pd);
+		if (r->rt)
+			pf_route6(m0, r, dir, kif->pfik_ifp, s, &pd);
+		break;
```

```
+     }

      return (action);
 }
Index: sys/net/pf_ioctl.c
===================================================================
RCS file: /cvs/src/sys/net/pf_ioctl.c,v
retrieving revision 1.211
diff -u -p -r1.211 pf_ioctl.c
--- sys/net/pf_ioctl.c     24 Nov 2008 13:22:09 -0000 1.211
+++ sys/net/pf_ioctl.c     4 Feb 2009 05:18:14 -0000
@@ -1491,7 +1491,7 @@ pfioctl(dev_t dev, u_long cmd, caddr_t a
                    s->kif->pfik_name)) {
 #if NPFSYNC > 0
                     /* don't send out individual delete messages */
-                    s->sync_flags = PFSTATE_NOSYNC;
+                    SET(s->state_flags, PFSTATE_NOSYNC);
 #endif
                     pf_unlink_state(s);
                     killed++;
@@ -1516,11 +1516,6 @@ pfioctl(dev_t dev, u_long cmd, caddr_t a
                if (psk->psk_pfcmp.creatorid == 0)
                     psk->psk_pfcmp.creatorid = pf_status.hostid;
                if ((s = pf_find_state_byid(&psk->psk_pfcmp))) {
-#if NPFSYNC > 0
-                    /* send immediate delete of state */
-                    pfsync_delete_state(s);
-                    s->sync_flags |= PFSTATE_NOSYNC;
-#endif
                     pf_unlink_state(s);
                     psk->psk_killed = 1;
                }
@@ -1566,11 +1561,6 @@ pfioctl(dev_t dev, u_long cmd, caddr_t a
                    !strcmp(psk->psk_label, s->rule.ptr->label))) &&
                    (!psk->psk_ifname[0] || !strcmp(psk->psk_ifname,
                    s->kif->pfik_name))) {
-#if NPFSYNC > 0
-                    /* send immediate delete of state */
-                    pfsync_delete_state(s);
-                    s->sync_flags |= PFSTATE_NOSYNC;
-#endif
                     pf_unlink_state(s);
                     killed++;
                }
Index: sys/net/pfvar.h
===================================================================
RCS file: /cvs/src/sys/net/pfvar.h,v
retrieving revision 1.282
diff -u -p -r1.282 pfvar.h
--- sys/net/pfvar.h   29 Jan 2009 15:12:28 -0000 1.282
+++ sys/net/pfvar.h   4 Feb 2009 05:18:14 -0000
@@ -59,7 +59,7 @@ struct ip6_hdr;

 enum { PF_INOUT, PF_IN, PF_OUT };
 enum { PF_PASS, PF_DROP, PF_SCRUB, PF_NOSCRUB, PF_NAT, PF_NONAT,
-       PF_BINAT, PF_NOBINAT, PF_RDR, PF_NORDR, PF_SYNPROXY_DROP };
```

```
+       PF_BINAT, PF_NOBINAT, PF_RDR, PF_NORDR, PF_SYNPROXY_DROP,
PF_DEFER };
 enum { PF_RULESET_SCRUB, PF_RULESET_FILTER, PF_RULESET_NAT,
        PF_RULESET_BINAT, PF_RULESET_RDR, PF_RULESET_MAX };
 enum { PF_OP_NONE, PF_OP_IRG, PF_OP_EQ, PF_OP_NE, PF_OP_LT,
@@ -742,6 +742,7 @@ struct pf_state {
     u_int8_t         direction;
     u_int8_t         pad[3];

+    TAILQ_ENTRY(pf_state) sync_list;
     TAILQ_ENTRY(pf_state) entry_list;
     RB_ENTRY(pf_state)    entry_id;
     struct pf_state_peer  src;
@@ -766,11 +767,14 @@ struct pf_state {
 #define   PFSTATE_ALLOWOPTS    0x01
 #define   PFSTATE_SLOPPY       0x02
 #define   PFSTATE_PFLOW        0x04
+#define   PFSTATE_NOSYNC       0x08
+#define   PFSTATE_ACK          0x10
     u_int8_t         timeout;
-    u_int8_t         sync_flags;
-#define   PFSTATE_NOSYNC       0x01
-#define   PFSTATE_FROMSYNC0x02
-#define   PFSTATE_STALE        0x04
+    u_int8_t         sync_state; /* PFSYNC_S_x */
+
+    /* XXX */
+    u_int8_t         sync_updates;
+    u_int8_t        _tail[3];
 };

 /*
@@ -827,8 +831,6 @@ struct pfsync_state {
     u_int8_t    updates;
 } __packed;

-#define PFSYNC_FLAG_COMPRESS   0x01
-#define PFSYNC_FLAG_STALE 0x02
 #define PFSYNC_FLAG_SRCNODE    0x04
 #define PFSYNC_FLAG_NATSRCNODE 0x08

@@ -1657,6 +1659,7 @@ void pf_change_a(void *, u_int16_t *, u_
 int  pflog_packet(struct pfi_kif *, struct mbuf *, sa_family_t,
u_int8_t,
         u_int8_t, struct pf_rule *, struct pf_rule *, struct pf_ruleset
*,
         struct pf_pdesc *);
+void pf_send_deferred_syn(struct pf_state *);
 int  pf_match_addr(u_int8_t, struct pf_addr *, struct pf_addr *,
         struct pf_addr *, sa_family_t);
 int  pf_match_addr_range(struct pf_addr *, struct pf_addr *,
Index: sys/netinet/ip_ipsp.c
===================================================================
RCS file: /cvs/src/sys/netinet/ip_ipsp.c,v
retrieving revision 1.174
diff -u -p -r1.174 ip_ipsp.c
```

```
--- sys/netinet/ip_ipsp.c  22 Oct 2008 23:04:45 -0000 1.174
+++ sys/netinet/ip_ipsp.c  4 Feb 2009 05:18:14 -0000
@@ -38,6 +38,7 @@
  */

 #include "pf.h"
+#include "pfsync.h"

 #include <sys/param.h>
 #include <sys/mbuf.h>
@@ -52,6 +53,10 @@
 #include <net/pfvar.h>
 #endif

+#if NPFSYNC > 0
+#include <net/if_pfsync.h>
+#endif
+
 #ifdef INET
 #include <netinet/in.h>
 #include <netinet/in_systm.h>
@@ -788,6 +793,11 @@ tdb_free(struct tdb *tdbp)
          (*(tdbp->tdb_xform->xf_zeroize))(tdbp);
          tdbp->tdb_xform = NULL;
     }
+
+#if NPFSYNC > 0
+    /* Cleanup pfsync references */
+    pfsync_delete_tdb(tdbp);
+#endif

     /* Cleanup inp references. */
     for (inp = TAILQ_FIRST(&tdbp->tdb_inp_in); inp;
Index: sys/netinet/ip_ipsp.h
===================================================================
RCS file: /cvs/src/sys/netinet/ip_ipsp.h,v
retrieving revision 1.136
diff -u -p -r1.136 ip_ipsp.h
--- sys/netinet/ip_ipsp.h  8 Nov 2008 12:54:58 -0000  1.136
+++ sys/netinet/ip_ipsp.h  4 Feb 2009 05:18:14 -0000
@@ -303,6 +303,8 @@ struct tdb {               /* tunnel descriptor blo
 #define   TDBF_SKIPCRYPTO      0x08000    /* Skip actual crypto
processing */
 #define   TDBF_USEDTUNNEL      0x10000    /* Appended a tunnel header in
past */
 #define   TDBF_UDPENCAP        0x20000    /* UDP encapsulation */
+#define   TDBF_PFSYNC          0x40000    /* TDB will be synced */
+#define   TDBF_PFSYNC_RPL      0x80000    /* Replay counter should be
bumped */

     u_int32_t  tdb_flags; /* Flags related to this TDB */

@@ -342,6 +344,7 @@ struct tdb {               /* tunnel descriptor blo
     u_int8_t   tdb_sproto;     /* IPsec protocol */
     u_int8_t   tdb_wnd;   /* Replay window */
     u_int8_t   tdb_satype;     /* SA type (RFC2367, PF_KEY) */
```

```
+    u_int8_t   tdb_updates;    /* pfsync update counter */

     union sockaddr_union  tdb_dst;   /* Destination address */
     union sockaddr_union  tdb_src;   /* Source address */
@@ -375,6 +378,7 @@ struct tdb {                /* tunnel descriptor blo
     TAILQ_HEAD(tdb_inp_head_in, inpcb)    tdb_inp_in;
     TAILQ_HEAD(tdb_inp_head_out, inpcb)   tdb_inp_out;
     TAILQ_HEAD(tdb_policy_head, ipsec_policy)  tdb_policy_head;
+    TAILQ_ENTRY(tdb)tdb_sync_entry;
 };

 struct tdb_ident {
Index: usr.sbin/tcpdump/print-pfsync.c
===================================================================
RCS file: /cvs/src/usr.sbin/tcpdump/print-pfsync.c,v
retrieving revision 1.32
diff -u -p -r1.32 print-pfsync.c
--- usr.sbin/tcpdump/print-pfsync.c   7 Oct 2007 16:41:05 -0000  1.32
+++ usr.sbin/tcpdump/print-pfsync.c   4 Feb 2009 05:18:18 -0000
@@ -62,9 +62,7 @@ struct rtentry;
 #include "pfctl_parser.h"
 #include "pfctl.h"

-const char *pfsync_acts[] = { PFSYNC_ACTIONS };
-
-void pfsync_print(struct pfsync_header *, int);
+void pfsync_print(struct pfsync_header *, const u_char *, int);

 void
 pfsync_if_print(u_char *user, const struct pcap_pkthdr *h,
@@ -80,6 +78,7 @@ pfsync_if_print(u_char *user, const stru
     }

     pfsync_print((struct pfsync_header *)p,
+        p + sizeof(struct pfsync_header),
         caplen - sizeof(struct pfsync_header));
 out:
     if (xflag) {
@@ -103,31 +102,57 @@ pfsync_ip_print(const u_char *bp, u_int
     if (len < PFSYNC_HDRLEN)
          printf("[|pfsync]");
     else
-         pfsync_print(hdr, (len - sizeof(struct pfsync_header)));
+         pfsync_print(hdr, bp + sizeof(struct pfsync_header),
+             len - sizeof(struct pfsync_header));
     putchar('\n');
 }

+const char *actnames[] = { PFSYNC_ACTIONS };
+
+struct pfsync_actions {
+    size_t len;
+    int (*print)(int, const void *);
+};
+
+int  pfsync_print_clr(int, const void *);
```

```
+int  pfsync_print_state(int, const void *);
+int  pfsync_print_ins_ack(int, const void *);
+int  pfsync_print_upd_c(int, const void *);
+int  pfsync_print_upd_req(int, const void *);
+int  pfsync_print_del_c(int, const void *);
+int  pfsync_print_bus(int, const void *);
+int  pfsync_print_tdb(int, const void *);
+int  pfsync_print_eof(int, const void *);
+
+struct pfsync_actions actions[] = {
+    { sizeof(struct pfsync_clr),         pfsync_print_clr },
+    { sizeof(struct pfsync_state),       pfsync_print_state },
+    { sizeof(struct pfsync_ins_ack),pfsync_print_ins_ack },
+    { sizeof(struct pfsync_state),       pfsync_print_state },
+    { sizeof(struct pfsync_upd_c),       pfsync_print_upd_c },
+    { sizeof(struct pfsync_upd_req),pfsync_print_upd_req },
+    { sizeof(struct pfsync_state),       pfsync_print_state },
+    { sizeof(struct pfsync_del_c),       pfsync_print_del_c },
+    { 0,                        NULL },
+    { 0,                        NULL },
+    { sizeof(struct pfsync_bus),         pfsync_print_bus },
+    { sizeof(struct pfsync_tdb),         pfsync_print_tdb },
+    { sizeof(struct pfsync_eof),         pfsync_print_eof }
+};
+
 void
-pfsync_print(struct pfsync_header *hdr, int len)
+pfsync_print(struct pfsync_header *hdr, const u_char *bp, int len)
 {
-    struct pfsync_state *s;
-    struct pfsync_state_upd *u;
-    struct pfsync_state_del *d;
-    struct pfsync_state_clr *c;
-    struct pfsync_state_upd_req *r;
-    struct pfsync_state_bus *b;
-    struct pfsync_tdb *t;
-    int i, flags = 0, min, sec;
-    u_int64_t id;
+    struct pfsync_subheader *subh;
+    int count, plen, alen, flags = 0;
+    int i;
+
+    plen = ntohs(hdr->len);

     if (eflag)
-        printf("PFSYNCv%d count %d: ",
-            hdr->version, hdr->count);
+        printf("PFSYNCv%d len %d", hdr->version, plen);

-    if (hdr->action < PFSYNC_ACT_MAX)
-        printf("%s:", pfsync_acts[hdr->action]);
-    else
-        printf("%d?:", hdr->action);
+    plen -= sizeof(*hdr);

     if (vflag)
```

```
           flags |= PF_OPT_VERBOSE;
@@ -136,83 +161,171 @@ pfsync_print(struct pfsync_header *hdr,
      if (!nflag)
           flags |= PF_OPT_USEDNS;

-     switch (hdr->action) {
-     case PFSYNC_ACT_CLR:
-           if (sizeof(*c) <= len) {
-                c = (void *)((char *)hdr + PFSYNC_HDRLEN);
-                printf("\n\tcreatorid: %08x", htonl(c->creatorid));
-                if (c->ifname[0] != '\0')
-                      printf(" interface: %s", c->ifname);
+     while (plen > 0) {
+           if (len < sizeof(*subh))
+                break;
+
+           subh = (struct pfsync_subheader *)bp;
+           bp += sizeof(*subh);
+           len -= sizeof(*subh);
+           plen -= sizeof(*subh);
+
+           if (subh->action >= PFSYNC_ACT_MAX) {
+                printf("\n    act UNKNOWN id %d", subh->action);
+                return;
           }
-     case PFSYNC_ACT_INS:
-     case PFSYNC_ACT_UPD:
-     case PFSYNC_ACT_DEL:
-           for (i = 1, s = (void *)((char *)hdr + PFSYNC_HDRLEN);
-               i <= hdr->count && i * sizeof(*s) <= len; i++, s++) {
-
-                putchar('\n');
-                print_state(s, flags);
-                if (vflag > 1 && hdr->action == PFSYNC_ACT_UPD)
-                      printf(" updates: %d", s->updates);
-           }
-           break;
-     case PFSYNC_ACT_UPD_C:
-           for (i = 1, u = (void *)((char *)hdr + PFSYNC_HDRLEN);
-               i <= hdr->count && i * sizeof(*u) <= len; i++, u++) {
-                bcopy(&u->id, &id, sizeof(id));
-                printf("\n\tid: %016llx creatorid: %08x",
-                    betoh64(id), ntohl(u->creatorid));
-                if (vflag > 1)
-                      printf(" updates: %d", u->updates);
-           }
-           break;
-     case PFSYNC_ACT_DEL_C:
-           for (i = 1, d = (void *)((char *)hdr + PFSYNC_HDRLEN);
-               i <= hdr->count && i * sizeof(*d) <= len; i++, d++) {
-                bcopy(&d->id, &id, sizeof(id));
-                printf("\n\tid: %016llx creatorid: %08x",
-                    betoh64(id), ntohl(d->creatorid));
-           }
-           break;
-     case PFSYNC_ACT_UREQ:
```

```
-          for (i = 1, r = (void *)((char *)hdr + PFSYNC_HDRLEN);
-              i <= hdr->count && i * sizeof(*r) <= len; i++, r++) {
-                bcopy(&r->id, &id, sizeof(id));
-                printf("\n\tid: %016llx creatorid: %08x",
-                    betoh64(id), ntohl(r->creatorid));
+
+          count = ntohs(subh->count);
+          printf("\n    act %s count %d", actnames[subh->action],
count);
+          alen = actions[subh->action].len;
+
+          if (alen == 0) {
+               printf("\n    unimplemented action");
+               return;
           }
-          break;
-     case PFSYNC_ACT_BUS:
-          if (sizeof(*b) <= len) {
-               b = (void *)((char *)hdr + PFSYNC_HDRLEN);
-               printf("\n\tcreatorid: %08x", htonl(b->creatorid));
-               sec = b->endtime % 60;
-               b->endtime /= 60;
-               min = b->endtime % 60;
-               b->endtime /= 60;
-               printf(" age %.2u:%.2u:%.2u", b->endtime, min, sec);
-               switch (b->status) {
-               case PFSYNC_BUS_START:
-                    printf(" status: start");
-                    break;
-               case PFSYNC_BUS_END:
-                    printf(" status: end");
-                    break;
-               default:
-                    printf(" status: ?");
+
+          for (i = 0; i < count; i++) {
+               if (alen > len)
                    break;
-               }
+
+               if (actions[subh->action].print(flags, bp) != 0)
+                    return;
+
+               bp += alen;
+               len -= alen;
+               plen -= alen;
          }
+     }
+
+     if (plen > 0) {
+          printf("\n    ...");
+          return;
+     }
+     if (plen < 0) {
+          printf("\n    invalid header length");
+          return;
+	}
+	if (len > 0)
+		printf("\n	invalid packet length");
+}
+
+int
+pfsync_print_clr(int flags, const void *bp)
+{
+	const struct pfsync_clr *clr = bp;
+
+	printf("\n\tcreatorid: %08x", htonl(clr->creatorid));
+	if (clr->ifname[0] != '\0')
+		printf(" interface: %s", clr->ifname);
+
+	return (0);
+}
+
+int
+pfsync_print_state(int flags, const void *bp)
+{
+	struct pfsync_state *st = (struct pfsync_state *)bp;
+	putchar('\n');
+	print_state(st, flags);
+	return (0);
+}
+
+int
+pfsync_print_ins_ack(int flags, const void *bp)
+{
+	const struct pfsync_ins_ack *iack = bp;
+
+	printf("\n\tid: %016llx creatorid: %08x", betoh64(iack->id),
+	    ntohl(iack->creatorid));
+
+	return (0);
+}
+
+int
+pfsync_print_upd_c(int flags, const void *bp)
+{
+	const struct pfsync_upd_c *u = bp;
+
+	printf("\n\tid: %016llx creatorid: %08x", betoh64(u->id),
+	    ntohl(u->creatorid));
+
+	return (0);
+}
+
+int
+pfsync_print_upd_req(int flags, const void *bp)
+{
+	const struct pfsync_upd_req *ur = bp;
+
+	printf("\n\tid: %016llx creatorid: %08x", betoh64(ur->id),
+	    ntohl(ur->creatorid));
+
+	return (0);
+}
+
+int
+pfsync_print_del_c(int flags, const void *bp)
+{
+	const struct pfsync_del_c *d = bp;
+
+	printf("\n\tid: %016llx creatorid: %08x", betoh64(d->id),
+	    ntohl(d->creatorid));
+
+	return (0);
+}
+
+int
+pfsync_print_bus(int flags, const void *bp)
+{
+	const struct pfsync_bus *b = bp;
+	u_int32_t endtime;
+	int min, sec;
+	const char *status;
+
+	endtime = ntohl(b->endtime);
+	sec = endtime % 60;
+	endtime /= 60;
+	min = endtime % 60;
+	endtime /= 60;
+
+	switch (b->status) {
+	case PFSYNC_BUS_START:
+		status = "start";
 		break;
-	case PFSYNC_ACT_TDB_UPD:
-		for (i = 1, t = (void *)((char *)hdr + PFSYNC_HDRLEN);
-		    i <= hdr->count && i * sizeof(*t) <= len; i++, t++)
-			printf("\n\tspi: %08x rpl: %u cur_bytes: %llu",
-			    htonl(t->spi), htonl(t->rpl),
-			    betoh64(t->cur_bytes));
-			/* XXX add dst and sproto? */
+	case PFSYNC_BUS_END:
+		status = "end";
 		break;
 	default:
+		status = "UNKNOWN";
 		break;
 	}
+
+	printf("\n\tcreatorid: %08x age: %.2u:%.2u:%.2u status: %s",
+	    htonl(b->creatorid), endtime, min, sec, status);
+
+	return (0);
+}
+
+int
+pfsync_print_tdb(int flags, const void *bp)
+{
```

```
+	const struct pfsync_tdb *t = bp;
+
+	printf("\n\tspi: %08x rpl: %u cur_bytes: %llu",
+	    htonl(t->spi), htonl(t->rpl), betoh64(t->cur_bytes));
+
+	return (0);
+}
+
+int
+pfsync_print_eof(int flags, const void *bp)
+{
+	const struct pfsync_eof *eof = bp;
+	int i;
+
+	printf("\n\thmac: ");
+	for (i = 0; i < sizeof(eof->hmac); i++)
+		printf("%02x", eof->hmac[i]);
+
+	return (0);
 }
```